Departamento de Ingeniería Telemática

PROYECTO FIN DE CARRERA

MOVILIDAD DE SESIONES SOBRE PLATAFORMA IMS

Autor: Ismael Fernández Castellano

Tutor: María Calderón Pastor

Leganés, julio de 2012

Agradecimientos

A María, Víctor e Ignacio por darme la oportunidad de realizar este proyecto.

A mi compañera y futura esposa, Noemí, por ser la persona que ha estado a mi lado en los momentos más duros y me ha ayudado a superarlos.

A mis padres, Carmen y Nasta, si he conseguido llegar hasta aquí ha sido gracias a ellos y a su impagable esfuerzo.

Resumen

En este proyecto se propone e implementa de forma completa un servicio para IMS. Este servicio permite la movilidad de sesiones entre terminales de un mismo usuario sin necesitar la intervención directa del otro extremo con el que tiene establecida la sesión.

Se utilizará una implementación de código abierto como Core IMS, y un servidor de aplicaciones JAIN SLEE también de código abierto en el que se desplegará la aplicación que se desarrolle y que permita la movilidad de sesiones. La aplicación manejará señalización SIP para permitir la movilidad, realizando procedimientos de transferencia entre los terminales del usuario, y se ayudará de otro servidor que interactuará al nivel de los datos intercambiados en la sesión, y que también será desarrollado.

Se emularán los terminales utilizando generadores de tráfico SIP/RTP y *softphones* para distintas plataformas, y se probarán varios escenarios para validar el correcto funcionamiento del servicio.

Palabras clave: IMS, SIP, JAIN SLEE, movilidad, transferencias.

Abstract

This project proposes and implements in a full way an IMS service. This service allows session mobility between several terminals of the same user, without requiring the direct intervention of the other side with which the session is established.

An open source implementation of an IMS Core will be used, and also an open source JAIN SLEE application server, in which the application to develop that allows session mobility will be deployed. The application will handle SIP signaling to enable mobility, performing transfer procedures between the user terminals, and will be helped by another server that will interact at the level of the exchanged data in the session, and that will be developed too.

The terminals will be emulated using SIP/RTP traffic generators, and also softphones for different platforms. Several test cases will be tested in order to validate the correct behavior of the service.

Keywords: IMS, SIP, JAIN SLEE, mobility, transfers.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

En este primer capítulo de introducción se describirán la motivación y los hechos que llevan a plantearse el objetivo principal de este proyecto. A continuación se volverá sobre este objetivo principal, y los distintos subobjetivos a abordar para conseguirlo. Finalmente se presentará un esquema de la memoria, resumiendo qué es lo que contiene y se describe en cada uno de los distintos capítulos y apéndices que la componen.

1.1 Motivación del proyecto

Cada vez existe un mayor número de dispositivos que requieren de conectividad, sobre todo debido a la enorme proliferación de dispositivos móviles. El usuario desea estar conectado en todo momento y desde cualquiera de sus dispositivos, y poder acceder a todo tipo de servicios como realizar llamadas, ver vídeos, compartir fotografías…

La conectividad se consigue desde distintas redes de acceso, como pueden ser las distintas redes de telefonía móvil, accesos de banda ancha, redes locales… El usuario demanda cada vez una mayor capacidad en dichas redes, lo que lleva a una constante evolución por parte de los operadores de red, que se están guiando en muchos casos hacia la adopción de la arquitectura *IP Multimedia Subsystem* (IMS) [1].

La arquitectura IMS propone un núcleo de red basado en IP al que poder acceder desde cualquier tecnología de acceso, y donde poder ofrecer de manera flexible todo tipo de servicios, tanto por parte del operador como de terceros, y sobre los que el operador tendrá control y podrá garantizar una calidad de servicio o *QoS*. El protocolo que se utilizará para poder prestar los servicios será el protocolo de señalización SIP [5], que permite establecer sesiones multimedia en las que intercambiar todo tipo de datos.

Es en este contexto que se acaba de presentar donde nace la motivación del proyecto. Partiendo de que se tenga una arquitectura IMS, en donde un usuario pueda establecer sesiones multimedia como llamadas de voz, videollamadas, o incluso sesiones de *IP Television* (IPTV) desde múltiples dispositivos; un servicio de gran valor añadido para el usuario sería el poder transferir las sesiones que tenga activas entre sus distintos dispositivos. De esta manera podría tener establecida una videollamada desde un ordenador y transferirla a su dispositivo móvil, o podría estar viendo la televisión en su dispositivo móvil y pasar a verla en el televisor. Este es el concepto de movilidad, y es lo que se pretende llevar a cabo.

El protocolo SIP permite la movilidad directamente mediante procedimientos para la transferencia de sesiones. El problema que aquí se presenta es que los terminales que se usen deben implementar estos procedimientos, tanto los terminales del usuario que pretende llevar

a cabo la transferencia como, y aquí está el punto más delicado, el terminal o servicio con el que tiene establecida la sesión.

Si se pretende dar el servicio de movilidad a un usuario, se puede asumir que sus terminales puedan llevar a cabo los procedimientos SIP necesarios, o que el hecho de utilizar el servicio le pueda suponer el tener que disponer de dichos terminales. Lo que no se puede asumir es que el extremo con el que se quiera establecer una sesión soporte estos procedimientos de SIP. De hecho IMS permite el poder establecer una sesión de audio con un terminal que se encuentre en una red de circuitos, y que directamente no soportará SIP, y muy probablemente la *Gateway* de IMS que permita conectar con dicho terminal tampoco soportará los procedimientos SIP necesarios.

Debido a esto se plantea el ofrecer el servicio mediante un *Application Server* (AS) desplegado en IMS Lo que hará dicho AS será manejar la señalización SIP permitiendo que el usuario que tenga movilidad realice los procedimientos de transferencia, y sin que para ello sea necesario que intervenga el extremo con el que el usuario tenga establecida la sesión. De hecho, dicho extremo no será consciente de que se haya producido una transferencia.

En el proyecto se plantea ofrecer la movilidad mediante dos tipos de transferencias. El primer tipo serán las denominadas como transferencias de tipo *Pull*, en este tipo de transferencias el usuario indicará desde uno de sus terminales que quiere que se le transfiera a dicho terminal la sesión que mantiene en otro. Las segundas serán las transferencias de tipo *Push* en las que el usuario elige, ahora directamente desde el terminal en que tiene establecida una sesión, otro terminal al que desea transferirla.

Esta es por tanto la motivación del proyecto, dado el entorno que se presenta en el actual escenario de las telecomunicaciones, cubrir la muy posible necesidad de movilidad que puedan tener los usuarios.

1.2 Objetivos

El objetivo del proyecto es desarrollar de forma completa y validar el funcionamiento de un servicio para IMS. Se pretende que el servicio permita dar movilidad a los usuarios, de tal forma que estos puedan transferir las sesiones que tengan activas entre sus distintos dispositivos o terminales. La transferencia se hará de forma transparente para el otro extremo de la sesión, que en ningún momento intervendrá para realizarla ni será consciente de ella.

Para conseguir este objetivo último del proyecto habrá que ir consiguiendo diferentes subobjetivos, que permitirán llegar a la solución completa. Estos subobjetivos se enumerarán a continuación:

- Se necesita disponer de una plataforma IMS, y para ello se utilizará una plataforma de código abierto ya desarrollada. Aún así habrá que familiarizarse con dicha plataforma para poder desplegar el servicio de movilidad en ella, así como para permitir que se puedan registrar usuarios en la plataforma y que hagan uso del propio servicio.

- Habrá que desarrollar una aplicación que interactúe con la señalización SIP para poder proporcionar el servicio de movilidad. Esta aplicación se desplegará en una implementación de código abierto de un *Application Server* de JAIN SLEE. Habrá primeramente que definir los procedimientos para interactuar con la señalización SIP, y posteriormente entender los conceptos básicos de JAIN SLEE y de la implementación que se va a usar, para así poder desarrollar la aplicación.

- Será necesario desarrollar otro servidor que interactúe al nivel de los datos de usuario intercambiados durante la sesión para permitir que la movilidad sea transparente. Este servidor se controlará desde el *Application Server* del punto anterior, que es el que maneja la información de señalización de la sesión, por lo que será necesario definir y desarrollar un interfaz de comunicación entre ambas entidades.

- Habrá que usar terminales que permitan probar el servicio de movilidad desarrollado. No es un subobjetivo el crear un terminal, pero sí que lo será el buscar herramientas que nos permitan probar y validar la solución.

- Se buscará validar el funcionamiento mediante los posibles casos de uso, y además intentar medir en cierta forma el rendimiento de lo desarrollado.

- Como subobjetivo que será general también para todos los anteriores, se buscarán soluciones lo más estándares y modulares posibles para, por ejemplo: permitir que se pudiera probar el *Application Server* en otro entorno IMS, que a su vez la aplicación JAIN SLEE desarrollada sirviera para otro contenedor distinto al usado, o que cualquier terminal IMS pudiera usarse para probar la solución.

Se intentará hacer un desarrollo gradual, añadiendo y probando funcionalidades poco a poco, para llegar al objetivo último y principal de tener probada y validada una solución completa con todos sus componentes.

1.3 Contenido de la memoria

La memoria se encuentra dividida en ocho capítulos y tres apéndices, cuyos principales contenidos se describirán a continuación.

Este es el primer capítulo de la memoria, titulado “Introducción y objetivos”, donde se ha hecho una introducción al proyecto partiendo de cuál es su motivación, se han descrito el objetivo principal y los diferentes subobjetivos que permitirán conseguirlo, y se está describiendo ahora el contenido de esta memoria.

El segundo capítulo de la memoria, titulado “Estado del arte”, describirá de una manera breve las bases teóricas sobre las que se sustenta el proyecto.

En el tercer capítulo, cuyo título es “Estructura de la solución”, se hará una breve descripción de cuáles serán los componentes que conformen la solución del proyecto, así como de las distintas tecnologías o soluciones que se utilizarán.

Los capítulos cuarto y quinto, titulados “Mobility Manager. Establecimiento de sesiones y transparencia” y “Mobility Manager. Gestión de las transferencias de sesión” respectivamente, describirán cómo se ha desarrollado la aplicación JAIN SLEE que interactúa a nivel de señalización para permitir el servicio de movilidad.

El sexto capítulo, de título “Transparency Manager”, se centrará en este caso en describir cómo ha sido el desarrollo del servidor encargado de interaccionar a nivel de los datos de usuario.

En el séptimo capítulo, titulado “Pruebas”, se hará un repaso detallado de cómo se ha probado y validado la solución implementada, mostrando además el proceso de cómo ha sido el desarrollo gradual de la aplicación.

El octavo y último capítulo, cuyo título es “Conclusiones y trabajos futuros”, se centra en analizar el grado de cumplimiento de los subobjetivos y el objetivo principal descritos en este primer capítulo. Además resume los aspectos del proyecto que se podrían mejorar, así como posibles desarrollos futuros que podrían continuar el trabajo realizado.

El primero de los apéndices, titulado “Manual de instalación”, explica paso a paso como poder instalar todos los componentes que conforman la solución.

El segundo apéndice, de título “Manual de usuario”, complementa al anterior detallando cómo manejar los distintos componentes y poder probar y validar la solución.

El tercer y último de los apéndices, titulado “Presupuesto”, muestra el presupuesto que ha supuesto la realización de este proyecto, junto a una división en tareas y subtareas para el cálculo de los tiempos.

Capítulo 2

Estado del arte

Este capítulo servirá para situar en contexto el proyecto, al repasar las fuentes teóricas sobre las que se sustenta. En primer lugar se centrará en la arquitectura IMS, elegida como entorno para el desarrollo, pasando posteriormente al protocolo de señalización que se utiliza en dicha arquitectura, SIP, y que al mismo tiempo será la base para desarrollar el servicio de movilidad. Finalmente se hará un breve resumen de JAIN SLEE, la tecnología empleada en el *Application Server* que proporcionará el servicio.

2.1 IMS

Las siglas IMS atienden a *IP Multimedia Subsystem*[1][2]. La primera de las palabras es la más importante, pues lo que se propone con la arquitectura IMS es un núcleo de red completamente basado en IP.

De este modo lo que se pretende es prestar toda una serie de servicios de nueva generación mediante el establecimiento de sesiones multimedia, y que permitan comunicaciones de audio y vídeo, el envío de mensajes o imágenes, funciones de presencia, etc. Para el establecimiento de dichas sesiones se utilizará el protocolo de señalización sobre IP SIP, sobre el que se hablará en 2.2. Se permitirá acceder al núcleo IP de la arquitectura utilizando diferentes tecnologías de acceso, y al mismo tiempo la posibilidad de conexión con distintas redes.

La arquitectura IMS ofrece muchas ventajas, tanto para el operador de red como para proveedores de servicio. Permite ofrecer servicios tipo Internet, dado el núcleo IP, pero con la ventaja de poder negociar y garantizar una calidad de servicio o QoS para satisfacer al usuario, y pudiendo el operador controlar la autenticación de usuarios, su autorización para acceder a servicios, y definiendo también un marco para poder tarificar.

En definitiva, de cara al operador ofrece una forma flexible de proporcionar servicios y mantenerlos bajo su control. De cara al usuario, una forma de asegurar calidad y buena experiencia al usar servicios. Y por último, de cara a proveedores de servicios y contenidos, una forma estándar y también flexible de ofrecer sus servicios.

Por todas estas razones, IMS es la arquitectura elegida sobre la cual desplegar el servicio de movilidad que se desarrolla en este proyecto.

IMS engloba muchos aspectos y consideraciones, pero no es un objetivo de este proyecto el estudio en profundidad de la arquitectura y todas sus posibilidades, sino únicamente el desarrollo de un servicio para esta arquitectura, y que se probará en una implementación de evaluación ya existente. En el próximo subapartado se describirán brevemente los componentes principales de la arquitectura que aparecerán en la implementación de

evaluación, y a lo largo del proyecto se tratarán otra serie de aspectos acerca de IMS, como serán la provisión de usuarios, o la manera en que se enrutan los mensajes de SIP a la hora de prestar servicios.

2.1.1 Componentes

La figura muestra los componentes básicos de la arquitectura IMS que se utilizarán y que forman parte de la implementación de evaluación que se usará en el proyecto, a los que se puede denominar como Core IMS.

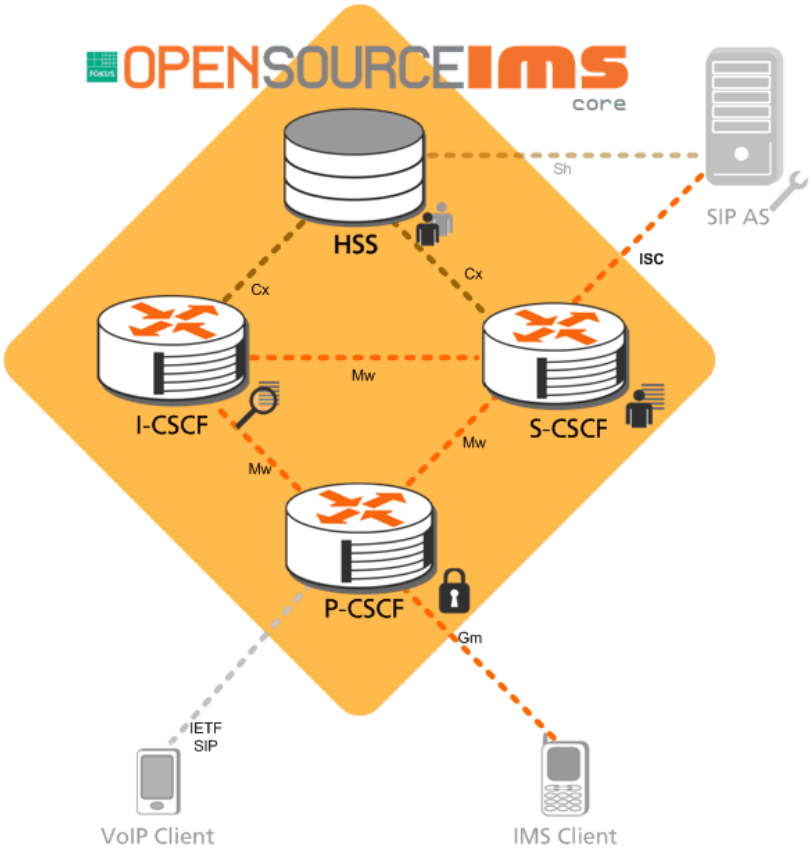

Figura 1: Componentes básicos de la arquitectura IMS[1]

El ***Home Subscriber Server (HSS)*** será la base de datos donde se guardará la información de los usuarios, así como de los posibles servicios a los que puedan acceder. Se almacenarán los datos necesarios para que los usuarios puedan registrarse, así como la información de los servicios que tengan activos y la forma de acceder a ellos. Se comunicará con el resto de los elementos utilizando el protocolo DIAMETER [3][4].

[1] Figura obtenida de [15].

Las entidades de control que manejarán la señalización SIP para el establecimiento de sesiones serán las denominadas como ***Call Session Control Function (CSCF)***. Habrá tres tipos distintos de entidades, que se enumerarán a continuación:

- ***Proxy CSCF (P-CSCF):*** Es el primer punto de contacto de los terminales con la arquitectura IMS. Toda la señalización SIP que los terminales intercambian pasa a través de esta entidad.
- ***Serving CSCF (S-CSCF):*** A cada usuario se le asignará un S-CSCF durante el proceso de registro. Será la entidad encargada de autenticar al usuario y controlar las sesiones que establezca, pudiendo dirigir las llamadas a distintos *Application Server* que proporcionen los servicios de los que dispone el usuario.
    - Se comunica con el HSS a través de la interfaz DIAMETER C_x para obtener información de autenticación y de los servicios de que dispone el usuario.
    - Interactúa con los *Application Server* a través de la interfaz SIP ISC.
- ***Interrogating CSCF (I-CSCF)***: Es el punto de entrada a la red IMS desde una red externa. Se comunica con el HSS a través de la interfaz DIAMETER C_x, y es la entidad responsable de asignar durante el proceso de registro el S-CSCF que dará servicio al usuario.

Para proporcionar servicios a los usuarios de IMS se utilizarán los ***Application Server (AS)***, que no forman parte del denominado Core IMS. Para la arquitectura de IMS los hay de tres tipos, pero se hará referencia en este documento únicamente al nativo para IMS, el SIP AS, que proporcionará servicios manejando señalización SIP. Un *Application Server* de este tipo será en el que se desplegará la aplicación de movilidad que se desarrolle en el proyecto.

El AS se comunicará por SIP con el S-CSCF a través de la interfaz ISC. Podrá comunicarse también con el HSS a través de la interfaz DIAMETER S_h.

2.2 SIP

SIP [5] es el protocolo de señalización que se utiliza en la arquitectura IMS. Se usa para el registro de usuario, y también para el establecimiento de sesiones multimedia, permitiendo proporcionar servicios como presencia, mensajería, o el que se propone en este proyecto, movilidad. En este subapartado se presentará el protocolo brevemente, orientando la exposición a la funcionalidad que se utilizará en el desarrollo del proyecto.

2.2.1 Entidades en SIP

SIP es un protocolo que utiliza un modelo cliente/servidor, y que está pensado para establecer sesiones extremo a extremo entre entidades conocidas como ***User Agent (UA)*** o agentes de usuario. Un agente de usuario puede actuar indistintamente como cliente (UAC) o como servidor (UAS). El UAC comienza mandando un tipo de mensajes SIP conocidos como

peticiones, y esperando recibir el otro tipo de mensajes, conocidos como repuestas. De forma complementaria el UAS espera recibir peticiones y responderlas. En el proyecto los UAs serán los terminales de usuario.

Los mensajes intercambiados entre dos UAs pueden atravesar una serie de servidores, que a su vez pueden ser de distintos tipos. Los que aparecerán en el proyecto a la hora de establecer, liberar y transferir sesiones serán los siguientes:

- ***Proxy:*** Un servidor proxy reencamina los mensajes y respuestas SIP entre los extremos de la comunicación. Pueden modificar los mensajes y las respuestas, pero acorde a unas estrictas normas recogidas en [5], que preservan la idea de señalización extremo a extremo. Pueden ser sin estado, si directamente reencaminan los mensajes sin guardar nada de información; o con estado si sí que guarda información, pudiendo por ejemplo retransmitir mensajes si no obtienen respuesta.
  El P-CSCF y el S-CSCF actuarán ambos como servidores proxy con estado en el establecimiento, liberación y transferencia de sesiones.
- ***Back To Back User Agent (B2BUA):*** Un B2BUA rompe la señalización extremo a extremo que caracteriza a SIP, para actuar como UAS de cara al UAC y viceversa. Puede utilizarse para prestar multitud de servicios, y en el caso del proyecto el *Application Server* se comportará como un B2BUA para prestar el servicio de movilidad a los usuarios.

2.2.2 Peticiones en SIP

Las peticiones básicas de SIP definidas en [5] son las siguientes:

- ***INVITE:*** Se utiliza para iniciar el establecimiento de sesiones multimedia.
- ***ACK:*** Confirma la recepción de una respuesta final para un establecimiento de sesión. Esta respuesta final puede significar tanto que se haya establecido de forma correcta la sesión como que no.
- ***CANCEL:*** Se utiliza para terminar intentos de establecimiento de sesión, cuando aún no se ha recibido una respuesta final para el INVITE.
- ***BYE:*** Para liberar sesiones ya establecidas.
- ***REGISTER:*** Se utiliza para registrar la información de contacto de un UA. Es lo que usarán los terminales de usuario en IMS para indicar dónde estarán localizables cuando se quiera establecer una sesión con ellos.
- ***OPTIONS:*** Para preguntar a un UA o un servidor acerca de sus capacidades.

Con estas peticiones se llevan a cabo los establecimientos y liberaciones de sesiones que se producen en el proyecto, así como el registro de los terminales de usuario en el Core IMS. Para prestar el servicio de movilidad, las transferencias de tipo *Pull* se realizarán con peticiones INVITE con una cabecera especial, que se verá en 2.2.4. Para las transferencias de tipo *Push* serán necesarias nuevas peticiones:

- ***REFER:*** Se define en [10], y de forma general un UA lo utiliza para indicar a otro UA que acceda a un determinado recurso. Cuando dicho recurso es una dirección de un terminal, lo que se está intentando es realizar una transferencia [9]. Este mensaje supone una suscripción implícita, mediante la cual el UA receptor informa al originante del estado. Para informar de dicho estado se utiliza la petición NOTIFY.
- ***NOTIFY:*** Se define en [11], y de forma general se utiliza para notificar a un UA de que se ha producido algún cambio o un evento en el que está interesado, pues previamente el UA se habrá suscrito para recibir estas notificaciones. Como se ha indicado anteriormente, en el proyecto se utilizarán para informar del estado de la transferencia.

Todas las peticiones SIP tienen en su línea inicial una dirección, denominada Request URI, que indica hacia dónde se dirigen dichas peticiones, y que se usará en el enrutamiento de las peticiones.

2.2.3 Respuestas en SIP

Las respuestas en SIP se agrupan en seis clases distintas. Constan de tres dígitos, el primero de los cuales hace referencia a la clase, mientras los dos siguientes especifican en mayor medida la respuesta. Así mismo se usa una frase textual acompañando al código, únicamente a efectos informativos y cara al usuario humano (por ejemplo para mostrase en un terminal para informar al usuario del estado de la respuesta), pues los servidores y terminales únicamente entienden los códigos numéricos. Las clases son las siguientes:

- ***1XX, Repuestas provisionales o informativas:*** Usadas para indicar que se está procesando una solicitud o establecimiento de sesión, y que preceden a una repuesta de otra familia, que se considera ya como final. Por ejemplo, la repuesta 180 Ringing, indica que se ha notificado en el terminal que se quiere establecer una sesión.
- ***2XX, Respuestas de éxito:*** Indican que el mensaje ha tenido éxito o se ha aceptado. Por ejemplo 200 OK para aceptar un establecimiento de sesión, o 202 Accepted [10] para aceptar un REFER.
- ***3XX, Respuestas de redirección:*** Por ejemplo 302 Moved Temporarily, que indica que el UA al que se intenta contactar se encuentra actualmente en otra localización.
- ***4XX, Error de cliente:*** El mensaje es erróneo, o la operación que se solicita no se puede completar. Por ejemplo, un 486 Busy Here indica que el llamado rechaza el establecimiento de sesión, o un 401 Unauthorized que es necesaria una autenticación para el registro de un usuario.
- ***5XX, Error de servidor:*** El servidor falla en atender una petición que a primera vista es correcta. Por ejemplo una respuesta 500 Server Internal Error, que indica que directamente se ha producido en error en el servidor al procesar la petición, como puede ser una excepción.
- ***6XX, Error global:*** La petición no se puede atender en ningún sitio. Por ejemplo está el mensaje 603 Decline, que implica que un cliente no desea aceptar sesiones.

2.2.4 Cabeceras principales

Se repasarán en este subapartado las cabeceras principales y que tendrán una mayor importancia en el desarrollo llevado a cabo en el proyecto.

- ***Call-ID:*** Contiene un identificador conocido como call-id. El call-id servirá para, junto con los tags de las cabeceras From y To que se verán a continuación, identificar un diálogo en SIP. Cada vez que se quiera iniciar una nueva sesión el UAC enviará un INVITE con un nuevo call-id. El B2BUA que se usará en el proyecto creará un INVITE con un nuevo call-id de tal forma que mantendrá dos diálogos SIP, uno con el UAC que inicia la sesión, y otro con el UAS que la recibe.
- ***From:*** Indica el UAC que origina una petición SIP. Contiene un tag, que crea el UAC con la primera petición, y que se mantiene tanto en peticiones posteriores como en las respuestas correspondientes al mismo diálogo. El B2BUA cambiará también este tag.
- ***To:*** Indica el UAS destino de una petición SIP. Contiene también un tag, que no aparece cuando el UAC manda la petición inicial, y que crea el UAS al generar la primera respuesta. Una vez creado se mantiene en las sucesivas respuestas y peticiones pertenecientes al diálogo. El tag lo cambiará también el B2BUA.
- ***Contact:*** Indica al destino la dirección en que quiere que se le contacte en futuras peticiones. En el INVITE inicial para el establecimiento de sesión el UAC indica dónde ser contactado en sucesivas peticiones del diálogo. A su vez en la respuesta 200 OK el UAS indica lo mismo. El valor aparecerá como Request URI en las sucesivas peticiones. El B2BUA del proyecto actualizará el valor de este cabecera, poniendo su dirección tanto en el INVITE inicial como en la respuesta 200 OK; de esta forma actuará como UAS para el UAC y viceversa.
- ***Via:*** En esta cabecera se indican todos los servidores por los que pasa una petición SIP hasta llegar al destino, así como el UAC origen. Es la cabecera que se utilizará para enrutar las respuestas, el UAS destino mandará la respuesta a la dirección que aparecezca en la cabecera Via superior; al recibir un servidor una repuesta eliminará la cabecera Via superior, que hace referencia a él, y enrutará la respuesta a la dirección que indique la siguiente cabecera Via.
- ***Record-Route:*** Cuando un servidor recibe una petición inicial de establecimiento de sesión dirigida hacia un destino, pondrá su dirección en una cabecera Record-Route si desea que el resto de peticiones pertenecientes al mismo diálogo pasen por él. Cuando la petición llegue al destino tendrá una serie de cabeceras Record-Route, que se mantendrán en las respuestas a dicha petición. Se usa por tanto para enrutar las peticiones. Mientras que el uso de la cabecera Via es obligatorio, el de Record-Route es opcional y lo usarán principalmente los servidores proxy que guarden estado. El P-CSCF y el S-CSCF utilizarán la cabecera Record-Route para crear las rutas por las que se enrutarán los mensajes a través del Core IMS. El B2BUA mantendrá rutas distintas en los dos diálogos SIP que maneje.
- ***Route:*** Complementa a la cabecera anterior para permitir la creación de rutas y el propio enrutamiento de las peticiones. Cuando el UAC recibe la respuesta con las cabeceras Record-Route, transforma estas cabeceras en cabeceras Route para las

siguientes peticiones que mande, de tal forma que sigan la ruta creada. Al igual que se hacía con las cabeceras Via, cuando un servidor recibe una petición, elimina la cabecera Route que le referencia. Mientras existan cabeceras Route la petición se enrutará utilizando dichas cabeceras, una vez ya no haya más cabeceras Route se enrutará hacia la Request URI.

Son necesarias dos cabeceras más para poder prestar el servicio de movilidad, que se detallan a continuación:

- ***Replaces:*** Se define en [6], y se utilizará para realizar transferencias de tipo *Pull* que permitan la movilidad. Irá en el INVITE que se envíe desde el nuevo terminal del usuario, y contendrá la información del diálogo SIP que se quiere sustituir, con el objetivo de que la sesión establecida se pueda transferir al nuevo terminal. La información estará compuesta por el call-id y los tags que identifican al diálogo SIP que se pretende sustituir.
- ***Refer-To:*** Esta cabecera se define en [10], y se utilizará en la petición REFER para indicar el recurso al que acceder. Se utilizará en las transferencias de tipo *Push* para indicar la dirección del nuevo terminal al que transferir la sesión.

2.2.5 Escenarios básicos de establecimiento y liberación de sesiones

Se presentan en este subapartado los diagramas básicos de señalización que muestran el establecimiento correcto y liberación de una sesión, así como fallos en el establecimiento de sesión debidos a que el usuario destino rechaza el establecimiento, o a que el usuario origen desiste de la petición. Estos escenarios son los que se contemplan en el proyecto, y se van a reflejar con los siguientes elementos y condiciones:

- El establecimiento de sesión se inicia desde el terminal de un usuario que tiene el servicio de movilidad.
- La sesión va dirigida al terminal de un usuario que no tiene movilidad.
- Aparece como entidad el *Application Server* que proporciona el servicio de movilidad, y que actuará como B2BUA.
- Aparece también el Core IMS, pero de forma genérica y sin especificar los distintos componentes por los que pasaría la señalización.

En primer lugar la Figura 2 muestra un establecimiento correcto de sesión, así como la liberación de dicha sesión desde el mismo terminal que empieza el establecimiento.

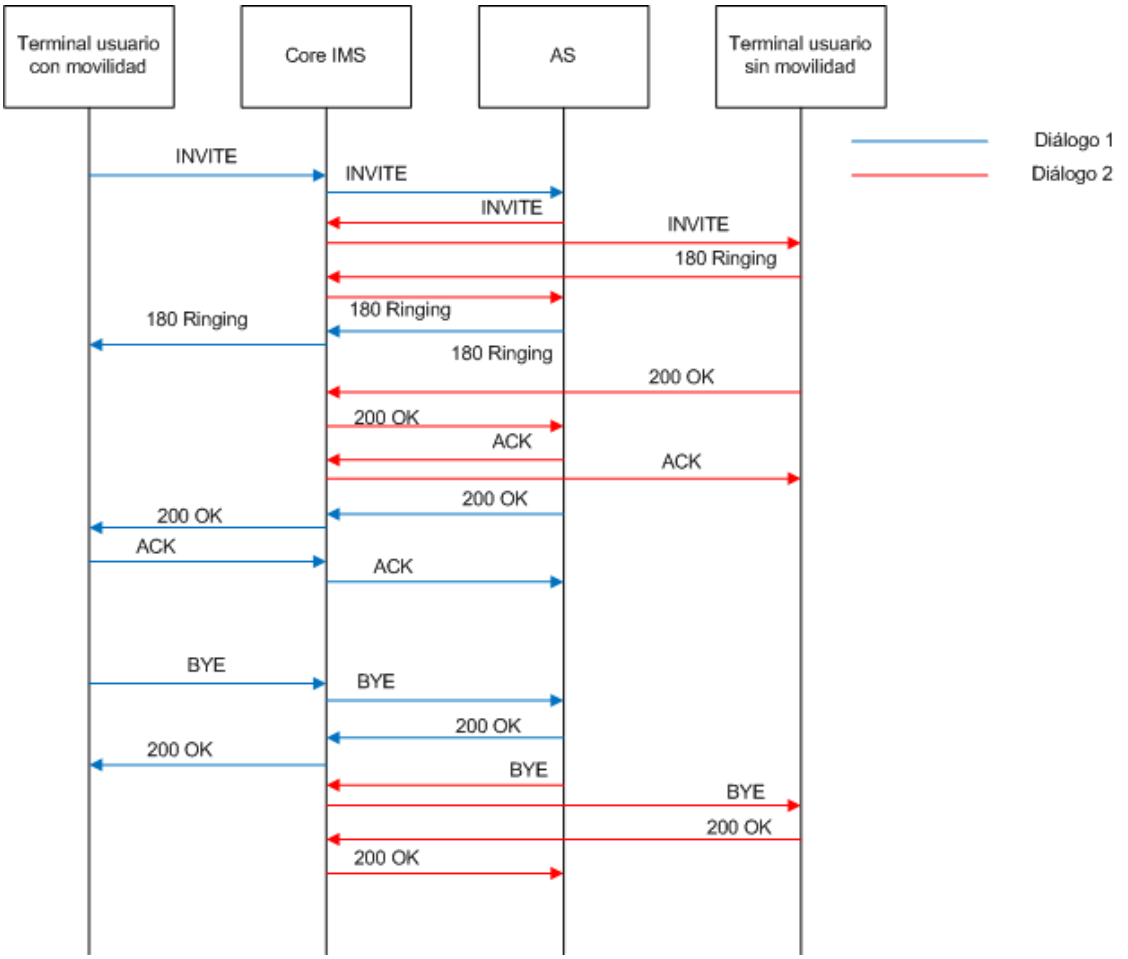

Figura 2: Establecimiento básico de sesión

La Figura 3 muestra el caso en que falla el establecimiento de sesión, debido a que el terminal del usuario destino lo rechaza.

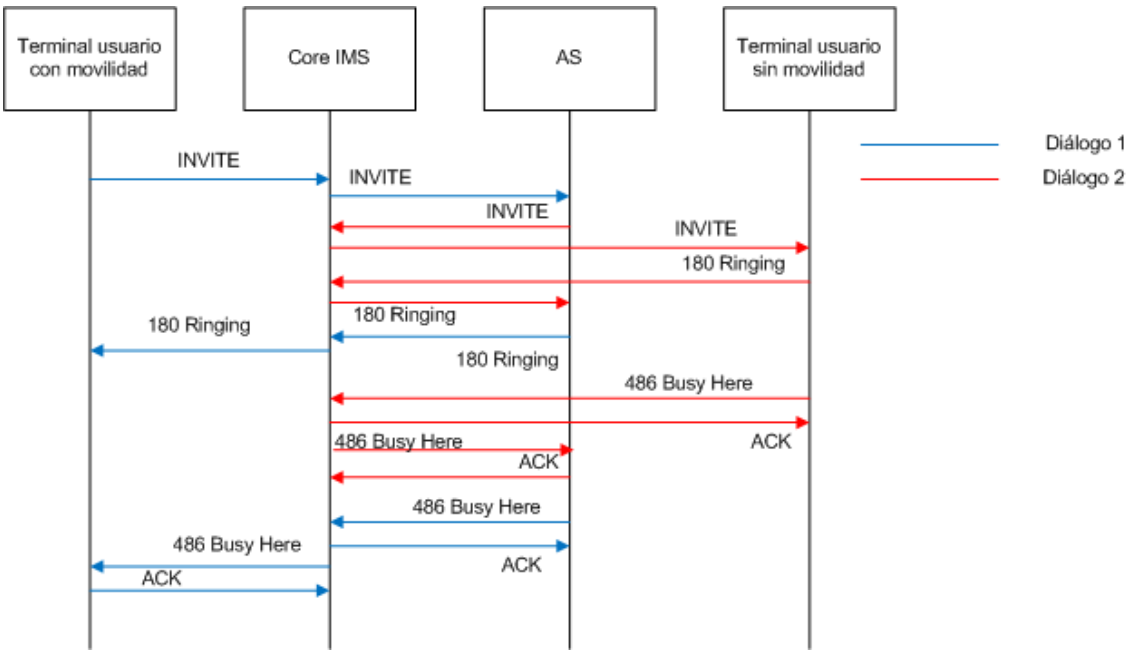

Figura 3: Fallo de establecimiento, el usuario destino rechaza

Por último, la Figura 4 muestra el escenario en que el terminal que inicia el establecimiento desiste.

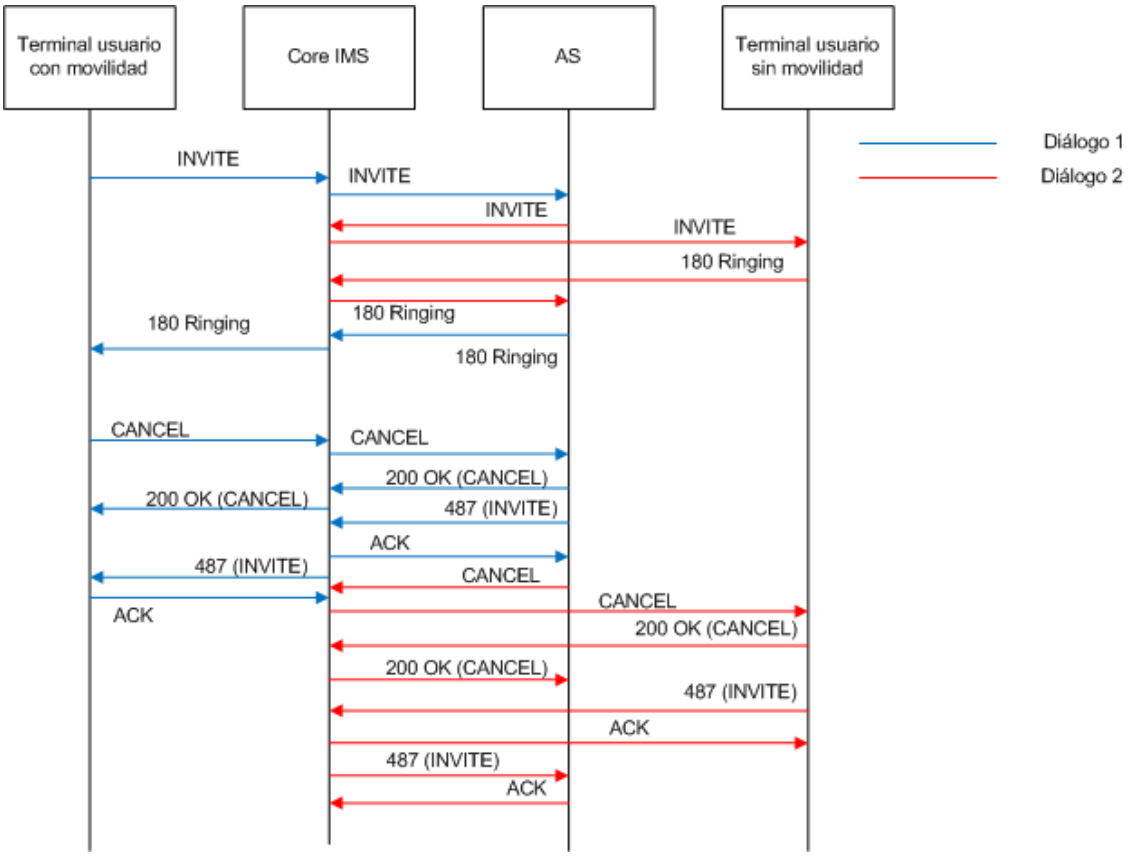

Figura 4: Fallo de establecimiento, el usuario origen desiste

Estos escenarios se irán desarrollando a lo largo del resto del documento.

2.2.6 Escenarios de movilidad propuestos

Este último subapartado muestra los diagramas de señalización que permiten la movilidad de sesiones tal y como se ha propuesto en el proyecto. Al igual que en los diagramas del subapartado anterior aparecerá el *Application Server* y de forma genérica el Core IMS, y se supone que el usuario que posee la movilidad es el que inicia el establecimiento de sesión. El inicio será siempre el mismo, el establecimiento correcto de una sesión que ya aparece en la Figura 2.

La Figura 5 muestra cómo se consigue la movilidad a través de una transferencia de tipo *Pull*. El usuario con movilidad establece una sesión y decide que le sea transferida a otro de sus terminales, enviando para ello una petición INVITE con la cabecera Replaces. El usuario con el que tiene establecida la sesión no es consciente de la transferencia, pues la realiza directamente el *Application Server*.

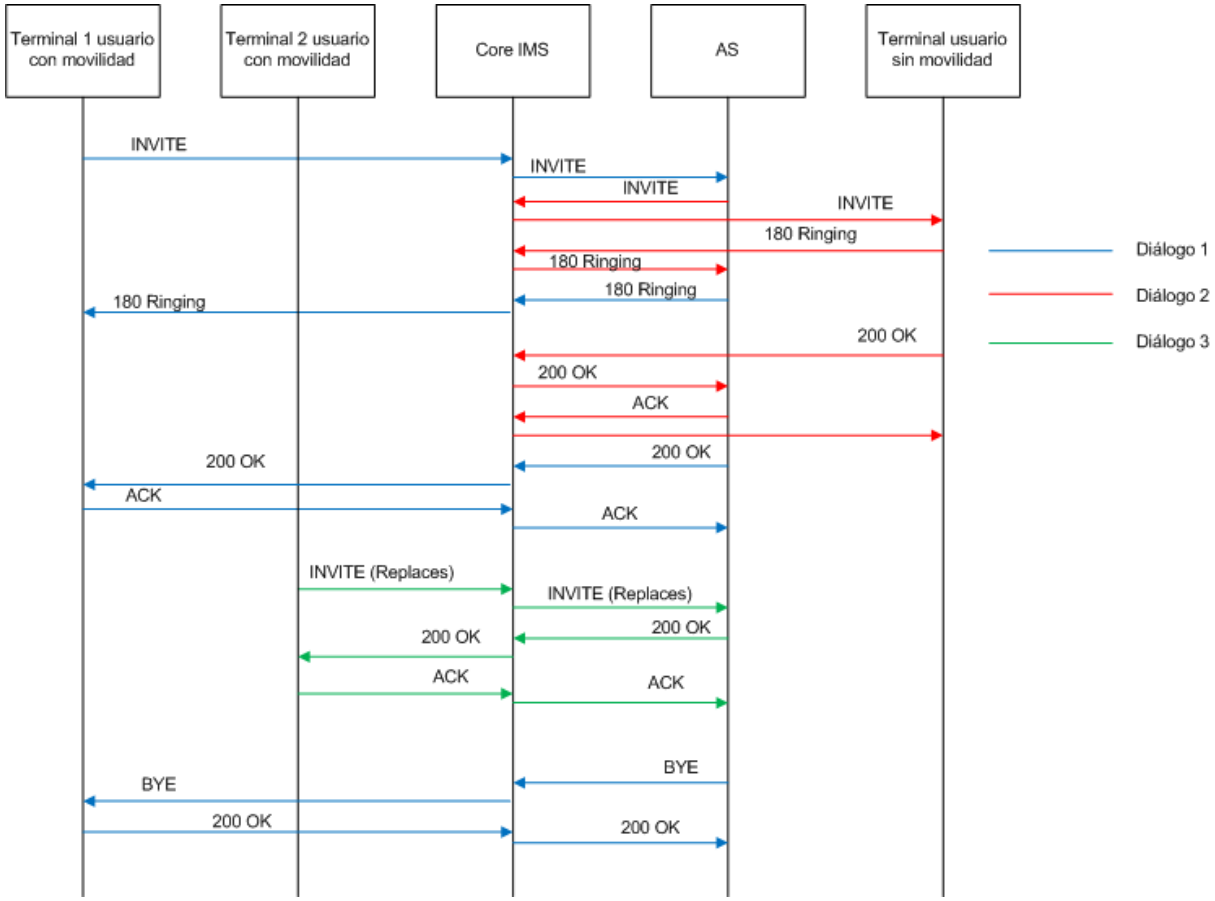

Figura 5: Movilidad de sesión mediante transferencia de tipo *Pull*

La Figura 6 muestra en este caso cómo se realiza una transferencia de tipo *Push* para conseguir la movilidad. En este caso el usuario indica, desde el terminal donde tiene establecida la sesión, que quiere que le sea transferida a otro de sus terminales; y para ello manda una petición REFER, indicando el nuevo terminal en la cabecera Refer-To. Nuevamente el proceso de transferencia lo realiza directamente el *Application Server*, sin la intervención del usuario con el que se tiene establecida la sesión.

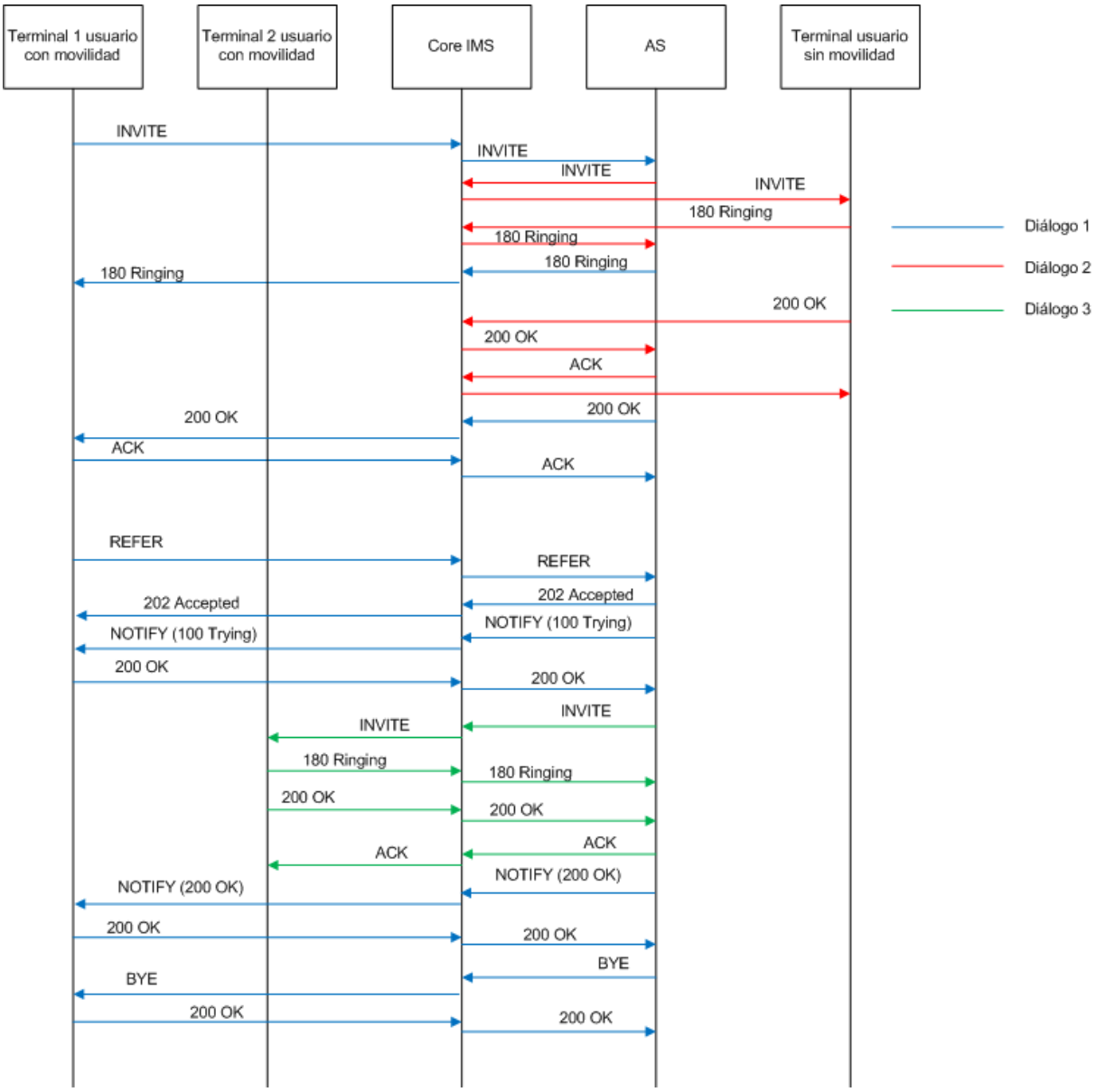

Figura 6: Movilidad de sesión mediante transferencia de tipo *Push*

Los dos tipos de transferencia que permiten la movilidad de sesiones se irán analizando con detalle a lo largo del documento, profundizando en sus distintos aspectos.

2.3 JAIN SLEE

JAIN SLEE [16] responde a las siglas de *Java APIs for Inteligent Networks Service Logic and Execution Environment*. Un SLEE es un concepto bien conocido en el mundo de las telecomunicaciones, y responde a un entorno para desarrollar aplicaciones orientadas al procesamiento de eventos, con baja latencia y alta tasa de transferencia. JAIN SLEE es el estándar de Java para un SLEE, y es a su vez el nodo central y parte más importante de la iniciativa JAIN [17].

La iniciativa JAIN define una serie de APIs que permiten el rápido desarrollo de productos y servicios de comunicaciones de nueva generación basados en Java, con la ventaja de permitir la portabilidad de servicios, un desarrollo abierto, y la independencia del operador de red. Pretende hacer frente entre otras cosas al problema de las soluciones propietarias, cerradas para cada fabricante u operador de red, y que conllevan tiempos y gastos innecesarios a la hora de crear y actualizar servicios debido a las necesidades de un desarrollo específico y a medida.

Volviendo a JAIN SLEE la especificación define un modelo de componentes reutilizables que permite, mediante su composición, la creación de aplicaciones de comunicaciones de cualquier grado de complejidad. Define así mismo la forma que tendrán de relacionarse estos componentes con el propio contenedor donde estarán desplegados. Especifica todo lo necesario para desarrollar el SLEE, pero no impone ninguna forma de hacerlo; de esta forma permite que pueda haber múltiples implementaciones que cumplan el estándar, permitiendo la diversidad en el mercado, y que el desarrollo de las aplicaciones sea independiente de la implementación a usar.

Recogiendo de una manera más ordenada los principales objetivos de la especificación, estos serían:

- Definir una arquitectura de componentes estándar que permita el desarrollo de aplicaciones de comunicaciones orientadas a eventos.
- Permitir el desarrollo de aplicaciones pudiendo combinar componentes desarrollados con herramientas de distintos fabricantes.
- Permitir un desarrollo simple para las aplicaciones. El desarrollador no necesitará conocer detalles de bajo nivel, como puede ser la forma de gestionar diversos hilos de ejecución, tareas de las que se ocupará el propio contenedor.
- Adoptar la filosofía de Java *"Write Once, Run Anywhere"*. Los componentes se desarrollarán una única vez, y podrán utilizarse en cualquier plataforma conforme al estándar.
- Definir un marco común que permita adaptar al contenedor cualquier tipo de recursos, como pueden ser bases de datos o distintas torres de protocolos, de tal forma que el contenedor pueda interactuar también con el exterior.

La especificación JAIN SLEE es muy amplia y de una relativa complejidad. Los siguientes subapartados se centrarán en analizar brevemente sus componentes y conceptos principales, centrándose principalmente en los aspectos que se utilizan en el desarrollo del proyecto.

2.3.1 Servicio

Un servicio en JAIN SLEE estará compuesto por una serie de *Service Building Blocks* (SBBs). Uno de los SBBs se conocerá como el SBB raíz del servicio, y el resto de SBBs serán hijos de éste. Cuando se reciba en el SLEE un evento inicial para que se produzca una ejecución del servicio (como puede ser por ejemplo un mensaje INVITE de SIP que inicia una sesión), se instanciará

un nuevo SBB raíz para que procese dicho evento y se ejecute toda la lógica correspondiente al servicio.

2.3.2 Service Building Block (SBB)

Un *Service Builiding Block* (SBB) es el componente reusable que se utiliza en JAIN SLEE para la creación de servicios. Es un componente software con estado que encapsula lógica de servicio. El mismo componente puede utilizarse en más de un servicio. Un SBB define:

- Los tipos de evento que recibe y que genera.
- Una serie de campos denominados *Container Managed Persistent* (CMP), que le proporcionan un estado persistente.
- Métodos para el manejo de los distintos tipos de evento que el SBB puede recibir, así como métodos para los tipos de evento que puede generar.
- Una *SBB Local Interface* con métodos síncronos que pueden ser invocados desde otros SBBs.
- Los SBB hijos que puede tener.
- El estado que puede compartir con otros componentes mediante la definición de un *SBB Activity Context Interface* con una serie de atributos.

Se conoce como *SBB Entity* a una instancia de un SBB. Es una instancia lógica que representa el estado persistente del SBB. Una nueva *SBB Entity* se crea cuando:

- Se recibe un evento inicial para un servicio. El SLEE crea automáticamente una nueva *SBB Entity* y le entrega el evento.
- Durante la ejecución de un servicio, el SBB puede decidir crear una nueva *SBB Entity* de uno de sus SBB hijos para que participe en la lógica del servicio.

Un *SBB Object* es la clase Java que representa a la *SBB Entity*. El SLEE tiene un pool de *SBB Objects* que en tiempo de ejecución va utilizando para representar a las distintas *SBB Entity* que existan (según las distintas ejecuciones de servicio que se estén teniendo). Un *SBB Object* puede representar a lo largo del tiempo a distintas *SBB Entity*. El modo de funcionamiento es el siguiente:

- Cuando se crea una nueva *SBB Entity* según los casos vistos anteriormente, se asigna un *SBB Object* del pool para que la represente. Este *SBB Object* realizará la lógica correspondiente del servicio, que será tratar el evento inicial en el caso de una nueva ejecución del servicio, o realizar alguna tarea que el padre que lo acabe de crear le imponga (procesar el evento o ejecutar un método síncrono). Durante la ejecución de la lógica podrá actualizar el estado persistente, y una vez que termina la ejecución el *SBB Object* dejará de representar a la *SBB Entity* y volverá al pool.
- Cuando se recibe en el SLEE algún evento relacionado con la ejecución del servicio, el SLEE toma un *SBB Object* del pool y lo asigna a la *SBB Entity* responsable de dicha ejecución, cargando todo el estado persistente (por ejemplo, cuando se recibe algún

mensaje de señalización SIP perteneciente a una sesión que ya se está tratando). Al igual que en el caso anterior, el *SBB Object* ejecutará la lógica y volverá posteriormente al pool.

- Durante la ejecución de la lógica del servicio, el SBB puede crear nuevas *SBB Entity*, o también reclamar una *SBB Entity* que haya creado con antelación. Puede utilizar siempre la misma *SBB Entity* para realizar ciertas tareas del servicio, como puede ser la comunicación con un sistema externo, y cada vez que reclame la *SBB Entity* el SLEE obtendrá un SBB Object del pool para que la represente.

Un SBB puede utilizar las facilidades que proporciona el SLEE, como puede ser la *Timer Facility*, que permite definir temporizadores. Un SBB puede definir durante la ejecución de un servicio un temporizador (por ejemplo para controlar el tiempo que se puede esperar para que se confirme una sesión), y si dicho temporizador expirase lanzará un evento al SBB para que realice la lógica correspondiente.

Los eventos que puede recibir un SBB pueden venir de las facilidades del SLEE, como en el caso anterior, directamente de otro SBB, o del exterior mediante la utilización de *Resource Adaptors* (RAs), como es el caso de recibir mensajes de señalización SIP. De forma simétrica puede enviar eventos a otros SBBs, o hacia el exterior mediante el uso de RAs. La comunicación entre SBBs se realiza mediante este intercambio de eventos, o mediante la invocación de métodos síncronos del *SBB Local Interface*. Los SBBs también pueden compartir datos mediante el *SBB Activity Context Interface.*

2.3.3 Resource Adaptor (RA)

Para que el SLEE interactúe con recursos externos la especificación define los *Resource Adaptor* (RA), que son la forma de adaptar estos recursos externos para que puedan ser utilizados por los servicios (los SBBs en última instancia) en el SLEE.

La comunicación desde el recurso hacia el SLEE se modela como eventos que se reciben en los SBBs, y desde el SLEE hacia el exterior como métodos que se invocan en objetos que proporciona el propio RA.

Un *Resource Adaptor Type* define los tipos de evento que el RA enviará hacia el SLEE, y también las actividades en que disparará dichos eventos. Al mismo tiempo definirá objetos para que un SBB pueda iniciar nuevas actividades en el recurso externo, y también comunicarse directamente con él.

El *Resource Adaptor Type* es genérico, y pueden realizarse múltiples implementaciones de él que den lugar a distintos RA. Típicamente se estandarizará un *Resource Adaptor Type* y diferentes fabricantes proveerán su implementación, de la misma manera que ocurre con la propia especificación JAIN SLEE.

En último lugar, un *Resource Adaptor Entity* es una instancia lógica de un RA, y en un SLEE pueden crearse distintas *Resource Adaptor Entity* del mismo RA que tengan una configuración

diferente (por ejemplo lanzar dos pilas del mismo protocolo en distinto puerto). Las *SBB Entity* utilizarán estas *Resource Adaptor Entity* para comunicarse con los recursos externos.

2.3.4 Actividades

Una *Activity* o actividad representa un flujo de uno o más eventos, y estos eventos representan cambios que han sucedido en la actividad. Por ejemplo todos los mensajes relativos a la señalización SIP de una llamada, o de una transferencia. Las actividades son siempre externas al SLEE.

Un *Activity Object* es la representación en objeto Java de la actividad, y es lo que se define en los RAs.

Un *Activity Context* encapsula el *Activity Object* dentro del SLEE, representa el canal lógico a través del cual se emiten los eventos para los SBBs, y a su vez los SBBs lanzan los eventos. Un SBB se suscribe a un *Activity Context* para recibir los eventos que produzca la actividad subyacente, y se desuscribe si ya no está interesado en seguir recibiendo los eventos. Un SBB sólo recibirá los eventos de los *Activity Context* a los que esté suscrito. Al mismo tiempo, es en los *Activity Context* donde los SBBs guardan el estado que desean compartir con otros componentes.

Para resumir los distintos ámbitos:

- Una *Activity* pertenece al ámbito de los recursos externos, y es controlada por estos.
- Los *Activity Object* pertenecen y se controlan por los RAs.
- Un *Activity Context* pertenece al ámbito del SLEE, y se controla por éste.

Como último concepto, los SBBs interactúan con los *Activity Context* a través de objetos *Activity Context Interface* (ACI). Para poder acceder a atributos compartidos los SBBs deben definir unos objetos ACI específicos donde se indiquen los atributos, tal como se vio en 2.3.2.

2.3.5 Perfiles

Un *Profile* o perfil se utiliza para guardar datos. Se define un esquema para los datos que se guardarán, y una *Profile Table* contiene cero o más perfiles que contienen los datos acordes a dicho esquema. Lo último que se define es un *Profile Specification*, que contiene las interfaces necesarios para manejar los distintos perfiles. Los SBBs pueden interactuar con los perfiles en su lógica de servicio.

La aplicación desarrollada en el proyecto no hará uso de perfiles.

2.3.6 Librerías

La especificación JAIN SLEE define las librerías como unos componentes que proporcionan información común a otros componentes desplegados en el SLEE. Pueden usarse por ejemplo para definir clases comunes a más de un SBB.

2.3.7 Deployable Unit

Las *Deployable Units* (DUs), o unidades desplegables, son ficheros JAR que se utilizan para desplegar componentes en el SLEE. Pueden contener servicios, SBBs, eventos, perfiles, RAs y/o librerías.

Capítulo 3

Estructura de la solución

En este primer capítulo se describirá de forma general la solución propuesta e implementada en el proyecto, introduciendo cada uno de sus componentes.

Como ya se ha visto, el objetivo principal del proyecto es implementar y validar una solución que permita la movilidad de sesiones entre terminales de un mismo usuario en un entorno IMS. Esta movilidad se conseguirá mediante las ya descritas transferencias de tipo *Push* y *Pull* entre terminales simulados de un mismo usuario, siendo el procedimiento transparente para el otro extremo de la comunicación.

Se desarrollará una aplicación para desplegar en un *Application Server*, el denominado como ***Mobility Manager***, que se integrará dentro de un ***Core IMS*** de pruebas y que será el que preste el servicio de movilidad de sesiones. Lo hará interactuando con la señalización SIP que se intercambia entre los extremos de la comunicación a la hora de establecer, transferir y liberar sesiones. Para el manejo de los datos de usuario, que serán flujos RTP [13], el Mobility Manager se ayudará de un nuevo servidor que también se implementará, el denominado como ***Transparency Manager***.

Para probar y validar la implementación se simularán ***terminales*** IMS, utilizando para ello *softphones* y scripts generadores de tráfico SIP y RTP. La Figura 7 muestra el esquema de la solución, con los distintos componentes ya mencionados.

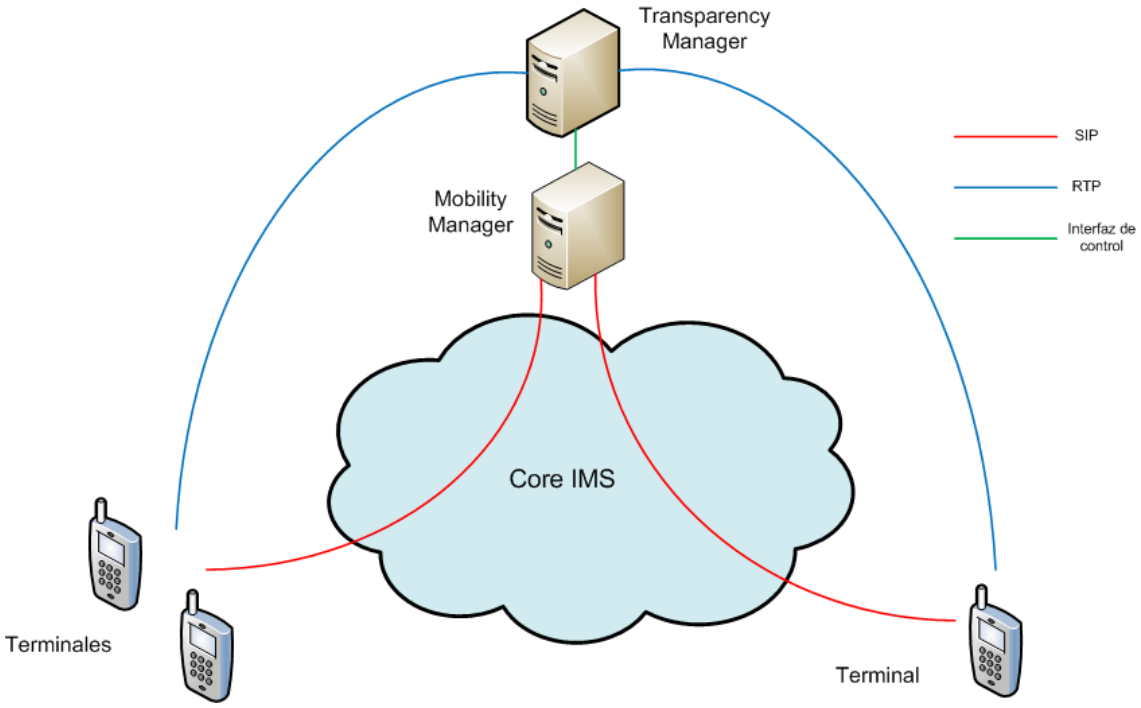

Figura 7: Esquema de la solución

3.1 Core IMS

Como Core IMS se utilizará la implementación de código abierto del instituto alemán FOKUS (*Fraunhofer Institute for Open Communication Systems*) [14], conocida como Open IMS Core [15]. Esta implementación consta de un *Home Subscriber Server* (HSS) para gestionar usuarios y servicios; y de los tres elementos encargados de controlar la señalización para el establecimiento de sesiones, los *Call Session Control Functions* (CSCFs): *Proxy* (P-CSCF), *Interrogating* (I-CSCF) y *Serving* (S-CSCF) [1], ya comentados en 2.1.1. Proporciona por tanto todo lo necesario para poder desplegar y probar servicios, lo que conforma el objetivo principal del proyecto.

La Figura 8 extraída directamente de la página web del proyecto muestra un esquema de los componentes proporcionados y previamente descritos. Es la misma Figura 1 que aparece en el apartado 2.1.1, que se vuelve a mostrar por completitud.

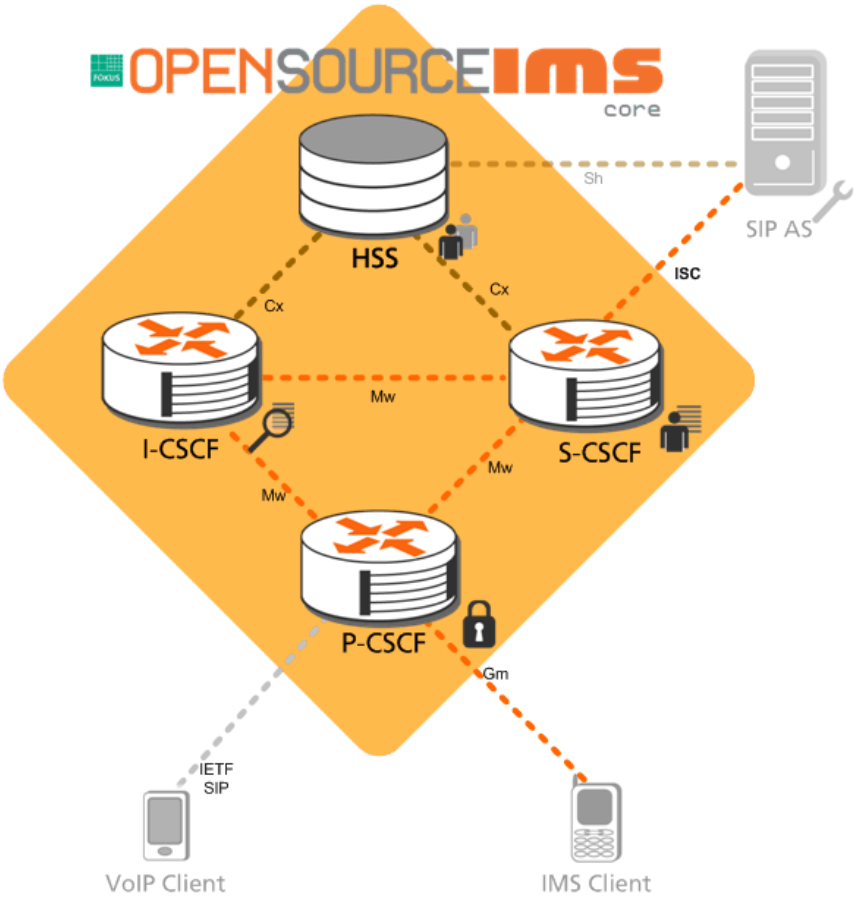

Figura 8: Open IMS Core

FOKUS proporciona todo lo necesario para instalar y configurar desde cero en una nueva máquina todo el entorno. Es un proceso de cierta complejidad, y con el objetivo de evitar dicha complejidad y posibles futuros problemas que pudieran surgir durante la instalación y configuración, y para en última instancia tener en un menor tiempo un entorno IMS operativo

en el que probar la solución del proyecto; se opta por utilizar una máquina virtual con todos los componentes, lista para ser utilizada nada más arrancar. Esta máquina virtual la proporciona el propio instituto FOKUS, con el mismo objetivo que se persigue de facilitar la labor de obtener un entorno operativo, y para arrancarla únicamente se necesita un programa gratuito como VMware Player [27].

Aun así, sí que será necesario un proceso de configuración en el Core IMS para tener todo lo que es ya el entorno del propio proyecto funcionando. Los pasos a realizar en la configuración son:

- Por defecto la dirección IP que se utiliza en la máquina virtual es la IP de loopback, 127.0.0.1. Utilizando esta IP sólo es posible registrar usuarios desde la propia máquina virtual, y los servidores de aplicaciones que se quieran desplegar deben estar también en dicha máquina. Para evitar esta serie de inconvenientes, y ya que la máquina virtual albergará únicamente el propio Core IMS, se utilizará una IP de red. El proceso de configuración se recoge en A.1.

- Será necesario dar de alta usuarios en el Core para poder probar el servicio de movilidad. Esto se hará desde la interfaz web de configuración del HSS, y el proceso queda reflejado en B.1.3.

- Hay que provisionar el servicio de movilidad. Para ello serán necesarios una serie de pasos, a realizar todos desde la interfaz web de configuración del HSS. A continuación se detallan los comunes a los dos tipos de transferencias permitidos, *Pull* y *Push*:
    - Dar de alta como *Application Server* al Mobility Manager, definiendo la IP y puerto en que recibirá la señalización SIP para poder manejar las sesiones. El Mobility Manager sólo manejara señalización SIP comunicándose con el Core IMS a través de la interfaz ISC, no se comunicará mediante DIAMETER con el HSS a través de la interfaz S_h. Se configurará el *Application Server* correspondiente de tal forma que si el Core IMS no puede contactarlo la llamada prosiga, lo que equivaldría a una llamada cursada de forma normal sin la posibilidad de tener movilidad.
    - Definir mediante un *Trigger Point* cómo se prestará el servicio de movilidad, cuándo se redirigirá el flujo de señalización SIP al Mobility Manager. Se hará bien cuando un INVITE sea originado desde un usuario con movilidad, o bien cuando el Invite se dirija al usuario con movilidad.
    - Asociar en un *Initial Filter Criteria* el *Trigger Point* y el *Application Server* anteriores.
    - Crear un *Service Profile* de movilidad, que contenga el *Initial Filter Criteria* anterior, y asociar el *Service Profile* a las identidades públicas de los usuarios con movilidad.

- En el caso de las transferencias de tipo *Push* el Mobility Manager necesitará enviar por sí mismo peticiones INVITE, tal y como se explicará en mayor detalle en 5.2.2. La configuración necesaria en el Core IMS para soportar este tipo de transferencias consistirá en:

- Definir una *Public Service Identity* (PSI) que represente al Mobility Manager, y que se usará cuando éste envíe peticiones INVITE a la hora de realizar transferencias de tipo *Push*.
- Es necesario modificar el Trigger Point explicado anteriormente, de tal forma que cuando el INVITE se dirija hacia un usuario con movilidad, se compruebe cuál es el origen del mensaje viendo el contenido de la cabecera P-Asserted-Identity. Si el INVITE procede del Mobility Manager significará que corresponde a un intento de transferencia de tipo *Push*, el Mobility Manager ya está interactuando y proporcionando movilidad, y no es necesario que este INVITE se enrute nuevamente hacia él. En el caso contrario de que no proceda del Mobility Manager, el INVITE deberá dirigirse hacia éste, de tal forma que se le permita la movilidad al usuario destino.

En B.1.2 aparecen una serie de figuras correspondientes a la configuración arriba mencionada que permitirá prestar el servicio de movilidad de sesiones.

3.2 Mobility Manager

El Mobility Manager será el *Application Server* que, integrado dentro del Core IMS, se encargará de proporcionar el servicio de movilidad de sesiones a los usuarios. Para ello se comportará como un *Back To Back User Agent* (B2BUA) de SIP [5] rompiendo la señalización SIP extremo a extremo, y controlará el Transparency Manager para el manejo de los flujos de datos intercambiados.

Se implementará como un servicio dentro de un servidor JAIN SLEE, que es la tecnología escogida por todas sus características y ventajas ya descritas en 2.3, principalmente el tratarse de un entorno orientado al procesamiento de eventos, con alta tasa de transferencia y baja latencia, ideal para cumplir los estrictos requisitos de aplicaciones de comunicaciones como las relacionadas con la señalización de red.

Se utilizará la implementación de Mobicents [18] de la especificación JAIN SLEE, la primera y única de código abierto. El equipo de Mobicents proporciona una serie de *Resource Adaptors* disponibles para usar, así como una serie de ejemplos que sirven de base para el desarrollo de servicios. Proporciona también una variada documentación, y cuenta con un activo grupo de desarrolladores y usuarios. Todos los anteriormente mencionados conforman los motivos de la elección de esta plataforma para el desarrollo.

El Capítulo 4 y el Capítulo 5 se centran en el desarrollo de este servicio para JAIN SLEE, que conforma en última instancia la parte central y más importante del proyecto. Tal y como se verá, los componentes de la especificación JAIN SLEE que conformen la aplicación serán:

- Un único servicio, el denominado Mobility Manager Service.
- 4 SBBs, cada uno encargado de una funcionalidad específica.
    - El principal, SBB raíz del servicio y padre de los demás, el B2BUA SBB. Encargado del establecimiento y liberación de la sesiones.

    - El Transparency SBB, encargado de interactuar con el Transparency Manager para el manejo de los datos.
    - Pull Transfer SBB, encargado de manejar las transferencias de tipo *Pull*.
    - Push Transfer SBB, encargado de manejar las transferencias de tipo *Push*.
- Una librería, que contendrá las clases necesarias para interactuar por RMI con el Transparency Manager, así como otra serie de clases comunes al resto de SBBs.
- El Resource Adaptor de SIP directamente proporcionado por Mobicents.

El desarrollo se ha hecho lo más conforme posible a la especificación JAIN SLEE, sin utilizar nada específico de la implementación de Mobicents, para así conseguir una solución que pueda ser utilizada en cualquier otra SLEE que implemente la especificación.

3.3 Transparency Manager

El Transparency Manager será el encargado de manejar los flujos de datos de usuario en las sesiones que se establezcan. Será una aplicación Java que se controlará desde el Mobility Manager mediante RMI, y que utilizará sockets para el manejo de los datos. Se explicará en profundidad en el Capítulo 6.

3.4 Terminales

Para poder probar la solución de movilidad propuesta e implementada se utilizan dos tipos de terminales: scripts de SIPp y *softphones* para IMS.

SIPp [20] es una herramienta de código abierto para generación y procesamiento de tráfico SIP. Mediante la edición de ficheros en XML permite definir distintos escenarios que reproduzcan el comportamiento que se quiere o se desea probar. De esta forma, a lo largo del desarrollo del proyecto se fueron creando escenarios que simulaban clientes y permitían probar el comportamiento del Mobility Manager durante el establecimiento, liberación y transferencias de sesiones.

Además de para probar el flujo de señalización, SIPp permite un manejo básico de flujos RTP, al ser capaz de enviar flujos y también de hacer un eco de los que recibe. De esta forma con esta herramienta se ha podido probar también el Transparency Manager, comprobando cómo los flujos de datos se redirigían correctamente entre los terminales a través de dicho servidor.

La Figura 9 muestra un ejemplo de ejecución de un script de SIPp, en el que se puede observar la señalización SIP que se espera durante el establecimiento de una sesión, y la transferencia de dicha sesión mediante el procedimiento denominado como *Push*. Las pruebas realizadas con SIPp aparecen en el Capítulo 7, su instalación en A.4.1, y su configuración y uso junto al resto de escenarios usados en B.4.1.

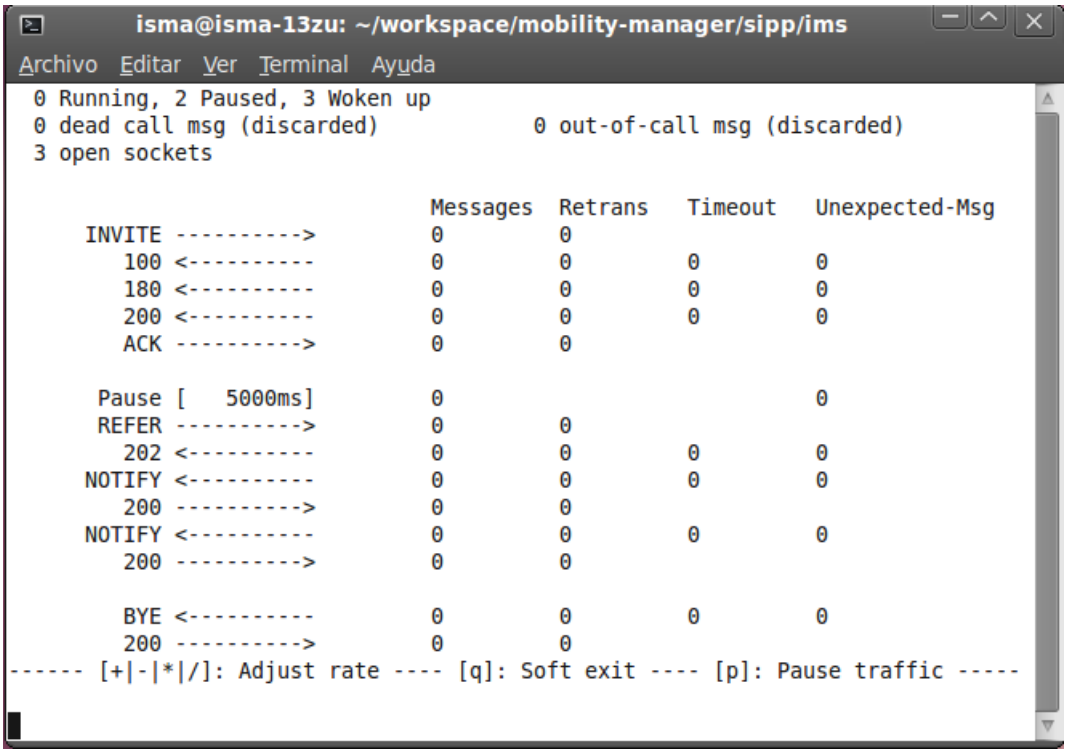

Figura 9: Ejemplo de script SIPp

Una vez probados los distintos escenarios que conforman la solución se emplean también *softphones* de IMS, que ya representan terminales más reales con los que poder probar y validar que la solución funciona. Se emplean la versión para Linux del softphone myMONSTER [21], también del instituto FOKUS, y el softphone para terminales Android IMSDroid [22].

Estos terminales proporcionan una forma más real y visual de probar la solución, pero también limitan los aspectos que se pueden probar, al no soportar ninguno de ellos la realización de transferencias de tipo *Pull*, y sólo poder realizar las transferencias de tipo *Push* con el terminal myMONSTER. Las pruebas realizadas se detallan en el Capítulo 7, mientras que la configuración y el uso de los softphones aparecen en A.4.2 y A.4.3, y B.4.2 y B.4.3 respectivamente.

La Figura 10 y la Figura 11 muestran el aspecto de los dos softphones de prueba.

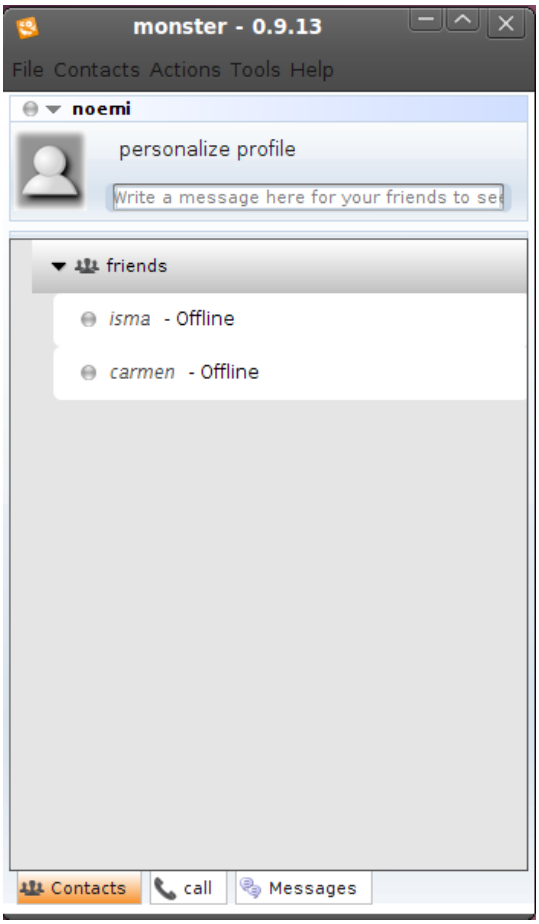

Figura 10: Softphone myMonster

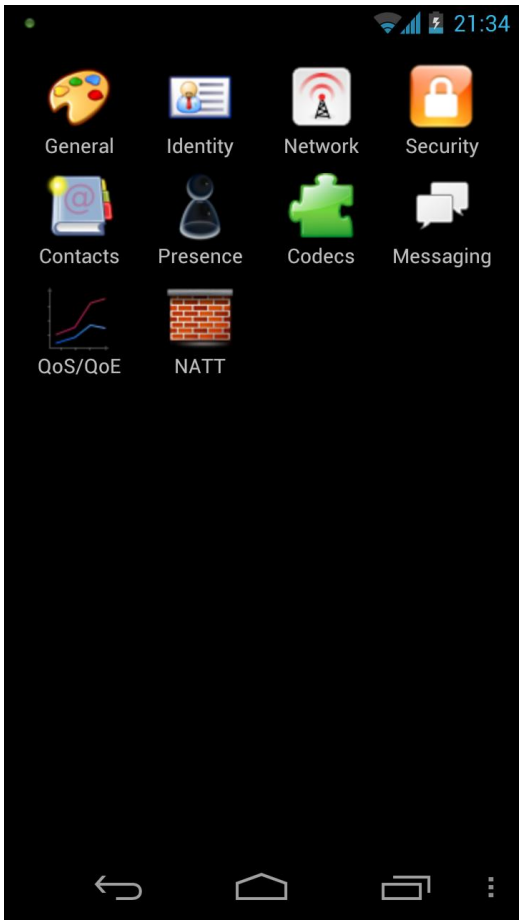

Figura 11: Softphone IMSDroid

Capítulo 4

Mobility Manager. Establecimiento de sesiones y transparencia

La implementación de este *Application Server* es la parte central y más importante en el desarrollo de este proyecto. Durante este capítulo y el siguiente se describirá de una forma detallada cómo es dicha implementación, analizando para ello las distintas partes que la componen. Es debido a la extensión por lo que se decide separar en dos capítulos

Desde el punto de vista de JAIN SLEE, tenemos un único servicio, denominado Mobility Manager Service. Cada nueva llamada que se reciba en el *Application Server*, y que será identificada por un INVITE originado desde un usuario con movilidad o dirigido a éste, disparará una nueva ejecución del servicio.

Este INVITE llega al *Application Server* porque es redirigido desde el Core IMS, y se recibe a través del *Resource Adaptor* (RA) de SIP que se encuentra desplegado. El servicio Mobility Manager Service tiene como SBB raíz el denominado B2BUA SBB, al que se entrega este INVITE en forma de evento desde el RA, iniciándose así una nueva ejecución del servicio.

Se explicará el funcionamiento partiendo desde este SBB Raíz, y a partir de él se irán describiendo el resto de componentes que conforman el servicio. La forma de ir describiendo los componentes seguirá al mismo tiempo lo que han sido las distintas fases del desarrollo de la aplicación.

4.1 B2BUA SBB

Es el SBB raíz del servicio, y al mismo tiempo la base de su implementación. Lo que hace este SBB es romper la señalización extremo a extremo de la comunicación, comportándose como un SIP *Back To Back User Agent* (B2BUA) [5], y permitiendo así que cualquiera de los extremos de la comunicación pueda moverse sin necesitar la intervención directa del otro.

Este SBB maneja el primer establecimiento básico de la sesión, el cual puede ser satisfactorio, o puede no serlo en caso de que el origen desista o el destino no acepte. En caso de que sea satisfactorio se comunicará con el Transparency SBB, que gestionará la transparencia de los datos tal y como se verá posteriormente. En caso de no serlo, finalizará directamente su ciclo de vida, finalizando así también la ejecución del servicio.

Posteriormente al establecimiento quedará a la espera de que se produzca la liberación de la sesión, que podrá ser iniciada por cualquiera de los extremos; una vez se produzca terminará su ciclo de vida, y al mismo tiempo finalizará la ejecución del servicio. Antes de que se produzca esta liberación pueden iniciarse transferencias, y en este caso el SBB instruirá a los SBBs Pull Transfer SBB y Push Transfer SBB para que las manejen según sea el caso. Estos SBB se analizarán en el siguiente capítulo.

Se observa por tanto que este SBB es el que orquesta todo el servicio, siendo su principal cometido el de gestionar el establecimiento y liberación de las sesiones, ayudándose del resto de componentes para llevar a cabo tanto la transparencia de los datos como las posibles transferencias de sesiones.

En este subapartado se analizará con mayor profundidad la función básica del SBB, el gestionar el establecimiento y liberación de las sesiones; mientras que las funciones relacionadas con la interacción con el resto de SBBs se citarán de manera breve, siendo analizadas más detenidamente cuando estos sean explicados. De esta forma se sigue el hilo de desarrollo de la aplicación, ya que el código del SBB fue ampliándose según se iban añadiendo las nuevas funcionalidades y desarrollando el resto de componentes.

4.1.1 Establecimiento correcto de sesión

El procedimiento comienza con la recepción de un INVITE. Este será el único evento que se declarará como inicial, y cada vez que se reciba uno se tendrá una nueva ejecución del servicio, creándose una nueva *SBB Entity* a la que se entregará el INVITE para que lo procese.

El tratamiento de este evento sienta las bases para que se tenga un comportamiento de B2BUA. Las acciones que se realizan son las siguientes:

- Se crea un diálogo a partir del INVITE recibido. A este diálogo se le denominará como diálogo entrante, y representará y manejará la interacción con el extremo que inicia la comunicación, el llamante. Será una de las patas del B2BUA.
- A partir del diálogo entrante se crea un nuevo diálogo, el denominado como diálogo saliente. Este nuevo diálogo será el que interactuará con el otro extremo de la comunicación, el llamado. Será la otra pata del B2BUA.
- A través de la facilidad correspondiente del RA de SIP se crearán *Activity Context Interface* (ACI) que representarán a cada uno de los diálogos. La *SBB Entity* se suscribirá a ambos, y así podrá recibir y procesar los eventos posteriores que puedan llegar de cada uno de los extremos, correspondientes a cada uno de los diálogos.
- Cada ACI guardará una referencia al otro diálogo, así se consigue tener relacionadas a nivel lógico las dos patas del B2BUA. Cuando se reciba un evento por una de las patas se tendrá acceso directo a la otra. Podemos referirnos a este diálogo como diálogo asociado.
- A partir del INVITE recibido se crea un nuevo INVITE, que será enviado por el diálogo saliente, iniciándolo de este modo.

El INVITE mantiene las direcciones en las cabeceras From y To, pero forma parte de un diálogo totalmente distinto al incluir un nuevo call-id y un nuevo tag local. Del mismo modo cambia el contenido de la cabecera Contact, para indicar que el B2BUA es el nuevo extremo de la comunicación.

Un punto importante aquí es saber enrutar este nuevo INVITE. Al estar integrado el *Application Server* en un Core IMS, el INVITE debe ser enrutado a través de éste, y acorde a los normas que

aquí se siguen. Como se vio en el capítulo anterior, el INVITE llegará al *Application Server* si viene originado desde un usuario que tiene activo el servicio de movilidad, o bien si va dirigido hacia éste, y lo hará desde el S-CSCF que sirva al usuario.

El S-CSCF incluirá una cabecera Route al enviar el INVITE al *Application Server*, indicando que el INVITE pueda volver a él, de tal forma que pueda seguir evaluando si se cumple alguna condición adicional y tiene que redirigir el INVITE a algún otro *Application Server* que preste servicio al usuario. Aquí el Mobility Manager funcionará como un Routeing B2BUA [2], manteniendo esta cabecera de tal forma que, aún iniciando un nuevo diálogo, el S-CSCF sepa que este nuevo diálogo está relacionado con el anterior, y que debe seguir evaluando criterios para el usuario, permitiendo así mantener la filosofía de encadenamiento de servicios de IMS.

LaTabla 1 muestra un ejemplo con los dos mensajes INVITE manejados en el Application Server, el recibido por la pata entrante, y el enviado por la saliente. En ella se pueden observar en negrita los cambios que se han ido indicando. Para esta y posteriores tablas relacionadas se tienen las siguientes condiciones:

- Se presta el servicio de movilidad al usuario llamante, isma@open-ims.test, y no al llamado, carmen@open-ims.test.
- Los terminales se encuentran en la IP 192.168.0.192, al igual que el Mobility Manager, y todos los componentes del Core IMS en la IP 192.168.0.193.

INVITE sip:carmen@open-ims.test SIP/2.0	INVITE sip:carmen@open-ims.test SIP/2.0
Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>,<sip:mo@pcscf.open-ims.test:4060;lr>	
Route: <sip:192.168.0.192:5060;lr>, <sip:iscmark@scscf.open-ims.test:6060;lr;s=1;h=0;d=0;a=7369703a69736d61406f70656e2d696d732e74657374>	**Route: <sip:iscmark@scscf.open-ims.test:6060;lr;s=1;h=0;d=0;a=7369703a69736d61406f70656e2d696d732e74657374>**
Call-ID: f3b3df97e6e9d0ff260c68053da9342d@192.168.0.192	**Call-ID: ca442e070a97f4f63c6a96de685a74ee@192.168.0.192**
CSeq: 9 INVITE	CSeq: 9 INVITE
From: <sip:isma@open-ims.test>;**tag=1006**	From: <sip:isma@open-ims.test>;**tag=4245c048**
To: <sip:carmen@open-ims.test>	To: <sip:carmen@open-ims.test>
Via: SIP/2.0/UDP 192.168.0.193:6060;branch=z9hG4bKbebf.cccc8a33.0	Via: SIP/2.0/UDP 192.168.0.192:5060;branch=z9hG4bK-353031-2dfaa231a3a6a86b0f4b0e698b037d60
Via: SIP/2.0/UDP 192.168.0.193:4060;branch=z9hG4bKbebf.40477467.0	
Via: SIP/2.0/UDP 192.168.0.192:5081;rport=5081;branch=z9hG4bKb8251ca432d918bb177009c4b46f3489383635	
Contact: "isma" <sip:isma@192.168.0.192:5081>	**Contact: <sip:192.168.0.192:5060;transport=udp>**
Max-Forwards: 15	Max-Forwards: 70
User-Agent: Fokus MONSTER Version: 0.9.13	User-Agent: Fokus MONSTER Version: 0.9.13
P-Asserted-Identity: <sip:isma@open-ims.test>	P-Asserted-Identity: <sip:isma@open-ims.test>
P-Charging-Vector: icid-value="P-CSCFabcd5017e0440000001c";icid-generated-at=192.168.0.193;orig-ioi="open-ims.test"	P-Charging-Vector: icid-value="P-CSCFabcd5017e0440000001c";icid-generated-at=192.168.0.193;orig-ioi=open-ims.test
Content-Type: application/sdp	Content-Type: application/sdp
Content-Length: 410	Content-Length: 410

Tabla 1: Invite modo B2BUA

Siguiendo con el establecimiento correcto de sesión se puede recibir de manera opcional una respuesta provisional de SIP, como es el 180 Ringing en caso de que el llamado lo envíe. Al procesar este evento se llama directamente a un método privado del SBB denominado

`forwardResponse`. Este método se invoca al recibir otras respuestas, tal y como se verá posteriormente, y su función es construir una nueva respuesta acorde a la recibida para ser enviada por el diálogo asociado, que se obtiene del ACI.

Por tanto en el 180 Ringing que se envía cambian el call-id y los tags que identifican el diálogo (en este caso también el tag remoto, que ya se tiene al ser una repuesta del llamado), y se actualiza también la información de la rutas que se mantienen en el diálogo. Las cambios son similares a los que aparecen en la Tabla 2, correspondiente a la siguiente respuesta recibida en un establecimiento correcto de sesión.

Esta siguiente y última respuesta será un 200 OK, que indica que el destino acepta la sesión. Al recibir el evento correspondiente a dicha respuesta las acciones realizadas en el SBB son:

- Se manda una petición ACK por el mismo diálogo por el que se recibe la respuesta. De esta forma quedaría confirmado el diálogo saliente y se evita el recibir retransmisiones de la respuesta. Este ACK sigue las rutas establecidas para el diálogo:
    - En la request URI irá el contenido de la cabecera Contact recibida en la respuesta, que identifica el extremo remoto de la comunicación.
    - Las cabeceras Record-Route recibidas en la respuesta irán como cabeceras Route en el ACK. De esta forma, el ACK se enruta de forma correcta por el Core IMS siguiendo el camino del INVITE.
- Se invoca al método `forwardResponse` descrito anteriormente, para mandar una respuesta 200 OK por el diálogo asociado y confirmarlo.

Además de los cambios comentados para la respuesta 180 Ringing, que se siguen manteniendo, es importante resaltar que en la cabecera Contact de la nueva respuesta aparecerá la dirección del B2BUA, pues éste será el extremo destino en el diálogo entrante y al que se redirigirán en última instancia el resto de posibles peticiones.

Respecto a las rutas establecidas en el diálogo, lo que se hace es cambiar las cabeceras Record-Route de la respuesta recibida por aquellas que se obtuvieron en la recepción del Invite. De esta forma, el resto de posibles peticiones provenientes del extremo origen seguirán el mismo camino que el Invite inicial, y acabarán en el *Application Server*. Este es el caso de la petición ACK, mensaje que finaliza el establecimiento correcto de la sesión, y que enviará el llamante una vez reciba la respuesta 200 OK.

La Tabla 2 muestra un ejemplo con las dos repuestas 200 OK manejadas por el B2BUA, donde se indican en negrita los cambios comentados. Éstas serían las respuestas correspondientes a los INVITE de ejemplo de la Tabla 1. Aparece a la izquierda la respuesta a enviar por el diálogo entrante, y a la derecha la recibida en el saliente.

SIP/2.0 200 OK **Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>,<sip:mo@pcscf.open-ims.test:4060;lr>** **Call-ID: f3b3df97e6e9d0ff260c68053da9342d@192.168.0.192** From: <sip:isma@open-ims.test>;**tag=1006** To: <sip:carmen@open-ims.test>;**tag=1729a286** CSeq: 9 INVITE Via: SIP/2.0/UDP 192.168.0.193:6060;branch=z9hG4bKbebf.cccc8a33.0,SIP/2.0/UDP 192.168.0.193:4060;branch=z9hG4bKbebf.40477467.0,SIP/2.0/UDP 192.168.0.192:5081;rport=5081;branch=z9hG4bKb8251ca432d918bb177009c4b46f3489383635 **Contact: <sip:192.168.0.192:5060;transport=udp>** P-Asserted-Identity: <sip:carmen@open-ims.test> Content-Type: application/sdp Content-Length: 229	SIP/2.0 200 OK **Record-Route: <sip:mt@pcscf.open-ims.test:4060;lr>,<sip:mt@scscf.open-ims.test:6060;lr>,<sip:mo@scscf.open-ims.test:6060;lr>** **Call-ID: ca442e070a97f4f63c6a96de685a74ee@192.168.0.192** From: <sip:isma@open-ims.test>;**tag=4245c048** To: <sip:carmen@open-ims.test>;**tag=1019** CSeq: 9 INVITE Via: SIP/2.0/UDP 192.168.0.193:4060;branch=z9hG4bK8fe8.d2bf6fa.0,SIP/2.0/UDP 192.168.0.193:6060;received=192.168.0.193;rport=6060;branch=z9hG4bK8fe8.6b194a76.0,SIP/2.0/UDP 192.168.0.193:6060;branch=z9hG4bK8fe8.5b194a76.0,SIP/2.0/UDP 192.168.0.192:5060;branch=z9hG4bK-353031-2dfaa231a3a6a86b0f4b0e698b037d60 **Contact: "carmen" <sip:carmen@192.168.0.192:5085>** Content-Type: application/sdp Content-Length: 229

Tabla 2: 200 OK modo B2BUA

Finalmente se recibirá por el diálogo entrante la request ACK ya comentada anteriormente, que confirmará dicho diálogo. El diálogo asociado se encuentra ya confirmado, tal y como se ha visto, por lo que no es necesario realizar ninguna acción adicional. Si se hubiera optado por no confirmar de manera inmediata el diálogo en la recepción del 200 OK habría que hacerlo ahora, accediendo al diálogo asociado a través del ACI, y construyendo y enviando una request ACK tal y como se vio anterioridad.

Queda finalizado el establecimiento correcto de la sesión, aunque cabe notar un detalle muy importante, se ha actuado únicamente a nivel de señalización. El B2BUA se encuentra en medio del camino de señalización, actuando como destino lógico para el llamante, y como origen lógico para el llamado, pero al no haber actuado sobre el contenido SDP de los mensajes ahora mismo el flujo de datos iría directo entre ambos extremos.

Lo que se quiere en el proyecto es controlar también este flujo de datos, de forma que si uno de los extremos se mueve, el flujo se le redirija automáticamente sin que tenga que intervenir el otro extremo. Esto se consigue con el Transparency Manager, que se controlará desde el Mobility Manager por medio del Transparency SBB descrito en 4.2.

4.1.2 Fallo en el establecimiento inicial de sesión

En este subapartado se parte del punto en que el *Application Server* ha enviado el INVITE que inicia el diálogo saliente. Aquí caben dos posibilidades para que no se produzca el establecimiento correcto visto en el subapartado anterior:

- Que el llamado, pudiendo haber enviado con anterioridad y de forma opcional la respuesta 180 Ringing, decida enviar una respuesta de error. Un caso bastante

habitual es que el llamado cuelgue, bien porque no está interesado en la sesión o porque no puede aceptarla en ese momento, con lo que enviaría un 486 Busy Here.
- Que el llamante desista en su intento de establecer la sesión, para lo cual enviará una petición CANCEL.

En el primero de los casos, se recibe como evento por el diálogo saliente la respuesta de error. Este mensaje se confirma de manera automática con un ACK en la pata saliente del B2BUA, labor que realiza directamente la pila SIP del RA. Con esto el diálogo saliente terminaría.

Se utiliza el método privado `forwardResponse` para construir una respuesta de error equivalente y mandarla al llamante por el diálogo asociado. De manera simétrica al otro diálogo se recibirá en este caso un ACK para confirmar la cancelación, que se queda en el RA y no se envía a la *SBB Entity*. Con esto terminaría el diálogo entrante.

Los cambios en la respuesta a enviar son los referidos a la identificación del diálogo y las rutas, tal y como se vio en el subapartado anterior y como se observa en la Tabla 2.

En el siguiente de los casos, al recibir un CANCEL del llamante, se actúa en las dos patas del B2BUA de la siguiente forma:

- En la pata entrante hay que aceptar el CANCEL. Para ello se usa directamente el método del RA que manda un 200 OK como respuesta al CANCEL y un 487 Request Terminated como respuesta al INVITE inicial. Así finalizaría el diálogo entrante.
- Para la pata saliente se obtiene el diálogo asociado y se manda un CANCEL para finalizarlo. De forma simétrica a la otra pata se recibirán una respuesta 200 OK al CANCEL, y una 487 Request Terminated al INVITE. Se comprueban estas respuestas pero no se tratan, ya se enviaron por la otra pata al aceptar el CANCEL.

En ambos casos la ejecución del servicio termina al terminarse los dos diálogos. Ya no se recibirán más eventos, las actividades que representan a los diálogos finalizan, y la *SBB Entity* al estar suscrita únicamente a estas dos actividades finalizará su ciclo de vida.

4.1.3 Liberación de la sesión

Habiéndose establecido la sesión tal y como se ha visto en el subapartado 4.1.1, llegará un momento en que cualquiera de los dos extremos deseará darla por concluida, y para ello mandará una petición BYE por el diálogo correspondiente.

Esta petición llegará al Application Server, pues su dirección será la que aparezca en la request URI sea cual sea el extremo que inicie la liberación de la sesión, y será enrutado de forma correcta a través del Core IMS. Esto es debido al funcionamiento en modo B2BUA que se ha visto, modificando la cabera Contact en el INVITE y en el 200 OK el *Application Server* se ha identificado como llamante para el llamado y viceversa, y se han mantenido de forma independiente las rutas para ambas patas del B2BUA.

El procedimiento una vez que la *SBB Entity* recibe el evento correspondiente al BYE es el siguiente:

- Construye una repuesta 200 OK para el BYE, y la manda por la misma pata del B2BUA por la que este último se ha recibido. De esta forma se da por finalizado el diálogo de esta pata.
- Se obtiene el diálogo asociado, correspondiente a la otra pata del B2BUA, y se construye y envía un BYE para cancelarlo, que seguirá las rutas establecidas.
- Se recibirá un 200 OK para el BYE enviado, con lo que el diálogo quedará finalizado. No es necesario tratarlo, pues ya se envió el mensaje equivalente por la otra pata nada más recibir el BYE.

No hace falta construir el BYE a enviar a partir del recibido, como se hizo con el INVITE, pues lo único que se necesita es que sea un BYE acorde al diálogo que se mantiene, que tenga sus identificadores de diálogo y siga las rutas establecidas.

La Tabla 3 muestra los mensajes BYE que permitirían cancelar las sesiones establecidas con los mensajes que aparecen en Tabla 1 y Tabla 2. En este caso quien inicia la liberación de la sesión es el llamado, el mensaje que éste envía y llega al B2BUA aparece a la derecha, mientras que en la izquierda se observa el mensaje enviado por el B2BUA.

BYE **sip:isma@192.168.0.192:5081** SIP/2.0	BYE **sip:192.168.0.192:5060;transport=udp** SIP/2.0
Via: SIP/2.0/UDP 192.168.0.192:5060;branch=z9hG4bK-353031-2e24b3ace4b90679900feb1ca3bd72e6	Via: SIP/2.0/UDP 192.168.0.193:6060;branch=z9hG4bK1fe8.84bb47b5.0 Via: SIP/2.0/UDP 192.168.0.193:6060;branch=z9hG4bK1fe8.74bb47b5.0 Via: SIP/2.0/UDP 192.168.0.193:4060;branch=z9hG4bK1fe8.fdb06c74.0 Via: SIP/2.0/UDP 192.168.0.192:5085;rport=5085;branch=z9hG4bK5310c8f355465abb777858ba63d42d31383631
Call-ID: f3b3df97e6e9d0ff260c68053da9342d@192.168.0.192	**Call-ID: ca442e070a97f4f63c6a96de685a74ee@192.168.0.192**
From: <sip:carmen@open-ims.test>;**tag=1729a286**	From: <sip:carmen@open-ims.test>;**tag=1019**
To: <sip:isma@open-ims.test>;**tag=1006**	To: <sip:isma@open-ims.test>;**tag=4245c048**
CSeq: 1 BYE	CSeq: 1 BYE
Max-Forwards: 70	Max-Forwards: 14
Route: <sip:mo@scscf.open-ims.test:6060;lr>,<sip:mo@pcscf.open-ims.test:4060;lr>	
	User-Agent: Fokus MONSTER Version: 0.9.13
	P-Asserted-Identity: <sip:carmen@open-ims.test>
Content-Length: 0	Content-Length: 0

Tabla 3: BYE modo B2BUA

Con este flujo de señalización ambos diálogos terminan, terminando las actividades que los representan. La *SBB Entity* finalizará su ciclo de vida al terminar estas dos actividades, ya no se recibirán más eventos.

Cabe notar que en el caso de que se produzcan transferencias de sesiones, los diálogos entrante y saliente del B2BUA cambiarán, tal y como se verá en el Capítulo 5. Lo importante para que se siga liberando de forma correcta la sesión es que el B2BUA SBB tenga siempre

actualizadas las referencias a los diálogos que conforman cada una de sus dos patas; consiguiendo esto los diálogos se cancelarán de forma correcta tal y como se ha visto, terminando la ejecución del servicio.

4.2 Transparency SBB

Este SBB será el encargado de comunicarse con el Transparency Manager para gestionar la transparencia de los datos. Es un SBB hijo del B2BUA SBB, durante el establecimiento de la sesión este último creará una instancia del Transparency SBB, la cual gestionará la transparencia de los datos correspondiente a dicha sesión.

Este subapartado analizará en detalle su funcionamiento, y lo hará en dos pasos. En primer lugar se verá cómo es la interacción con el Transparency Manager; y posteriormente se detallará cómo se ha implementado el SBB, y cómo es la comunicación con su SBB padre.

Este SBB manejará también la transparencia de los datos cuando se produzcan transferencias de sesiones. De hecho es necesaria su existencia para que estas transferencias se produzcan de forma transparente, esto es sin la intervención directa del extremo de la comunicación que no inicia el proceso de movilidad. En este caso habrá una interacción con los SBBs encargados de las transferencias, Pull Transfer SBB y Push Transfer SBB, se volverá sobre ello en el siguiente capítulo.

4.2.1 Interacción con el Transparency Manager

La decisión de separar de forma completa el Transparency Manager del Mobility Manager permite que ambas entidades puedan residir en máquinas independientes, y al mismo tiempo que pueda haber una separación total entre el camino que sigue la señalización y el que siguen los datos.

Es necesario que ambas entidades se comuniquen, y hay que decidir de qué forma hacerlo. Lo ideal desde el punto de vista del Mobility Manager, implementado en JAIN SLEE, sería una comunicación basada en un RA. Además del RA de SIP que ya se está usando, Mobicents proporciona otros que podrían servir, por ejemplo:

- Http-client: En caso de implementar un interfaz de comunicación basado en, por ejemplo, REST o Web Services, se podría utilizar este RA desde el Mobility Manager para hacer las invocaciones por HTTP al Transparency Manager.
- Diameter-base: Se podría implementar una interfaz basada en DIAMETER acorde a la comunicación que se necesita. Se usaría así un protocolo muy utilizado actualmente y que como se ha visto ya se emplea en otras partes de la solución, las interfaces del HSS con los CSCFs.

Ambas soluciones arriba planteadas serían una buena elección. Soluciones genéricas, basadas en estándares, y que permitirían implementar el Transparency Manager en cualquier tipo de lenguaje, siempre que se ofrezca la interfaz de comunicación definida. El problema es que ambas exigen un considerable esfuerzo en el desarrollo, y este desarrollo no es el principal foco u objetivo en la realización del proyecto.

La solución por la que se ha optado para conseguir contener los tiempos de desarrollo ha sido utilizar RMI, el mecanismo de invocación remota ofrecido como parte del estándar del lenguaje Java [19]. La solución es menos genérica, pues exige que ambas entidades estén desarrolladas en Java, pero no es un problema de fondo para el proyecto al ser Java el lenguaje de programación elegido desde el principio.

El Transparency Manager será un programa Java *stand-alone* que ofrecerá de manera remota un objeto con los métodos necesarios para controlar la transparencia de datos, y este objeto se utilizará desde el Mobility Manager. No es la forma más recomendable de invocar recursos remotos desde JAIN SLEE, siendo ésta el usar RAs tal y como se ha indicado, pero por las razones previamente esgrimidas es la opción elegida. El siguiente subapartado, 4.2.2, mostrará como en cierta forma se aísla este problema.

La interfaz remota que ofrece el Transparency Manager tiene los siguientes métodos, cuyo funcionamiento se explicará más detalladamente en el Capítulo 6:

- `SessionInfo addEarlySession(String id, SessionInfo userAgentSessionInfo)`: Mediante este método se le indican al Transparency Manager las direcciones IP y puertos que el llamante utilizará para comunicarse en una sesión identificada mediante un id único, y se espera a que éste responda con las direcciones IP y puertos que reservará para dicha sesión.
- `SessionInfo addConfirmedSession(String id, SessionInfo userAgentSessionInfo)`: En este caso se le comunican al Transparency Manager las direcciones IP y puertos del llamado, esperando que éste devuelva los que se van a usar para la sesión.
- `removeEarlySession(String id)`: Se le indica al Transparency Manager que la sesión no se ha establecido de forma correcta.
- `removeConfirmedSession(String id)`: Se le indica al Transparency Manager que la sesión ha finalizado de forma correcta.

El objeto `SessionInfo` guardará en un mapa las distintas direcciones IP y puertos que utilizarán los terminales de los clientes y el Transparency Manager para comunicarse, y cuyo número dependerá de los distintos medios que se empleen en la sesión: una IP y un puerto si la sesión es únicamente de voz, dos si es de voz y vídeo, etc. El tipo de medio servirá a su vez para indexar los valores del mapa.

Estos objetos, así como la propia interfaz remota, formarán parte de una librería JAIN SLEE que se desplegará junto al resto de componentes, y que usará el SBB para poder realizar las invocaciones. La interfaz remota tendrá un método más, el cual permitirá la transferencia de sesiones, y al que se hará referencia en el siguiente capítulo al analizar los SBBs encargados de la gestión de dichas transferencias.

4.2.2 Implementación e interacción con el B2BUA SBB

La tarea de comunicarse con el Transparency Manager mediante el uso de los métodos presentados en el subapartado anterior podría realizarla directamente el B2BUA SBB. El hecho de utilizar el Transparency SBB proporciona un nivel más de abstracción, permitiendo que el B2BUA SBB se ocupe únicamente de la señalización, mientras que el Transparency SBB lo hace de la transparencia de los datos.

Además permite que, si en un futuro, se cambiara la forma de comunicarse con el Transparency Manager (utilizando por ejemplo un RA), únicamente hubiera que modificar el Transparency SBB, o en su defecto diseñar uno nuevo. Se tienen SBBs centrados en la realización de tareas concretas y que además cumplen el objetivo de poder ser reutilizables, el Transparency SBB podría utilizarse en otro servicio que requiera transparencia de datos.

Yendo a la implementación el B2BUA SBB creará un SBB hijo de tipo Transparency SBB, una nueva *SBB Entity*, durante el establecimiento de la sesión, y guardará una referencia a él en un campo *Container Managed Persistent* (CMP). Posteriormente, y durante las distintas fases tanto del establecimiento de la sesión como de su liberación, utilizará esta *SBB Entity* del Transparency SBB para manejar la transparencia de los datos, siempre la misma.

La forma de utilizar el Transparency SBB es mediante invocaciones a métodos síncronos. Por tanto, se implementa para este SBB una *SBB Local Interface* donde se definen los métodos necesarios a invocar para conseguir la transparencia. Se detallan a continuación dichos métodos síncronos, indicando además en qué momentos de la gestión de la sesión vistos en el apartado 4.1 se invocarían.

- `SessionDescription getEarlySessionSDP(String mobilitySessionId, byte[] sdpBytes)`: En el establecimiento inicial de la sesión, y antes de que el primer INVITE sea enviado por el diálogo saliente, el B2BUA invoca este método síncrono del Transparency SBB.
  El B2BUA indica cuál es la carga SDP [12] recibida en el INVITE inicial, y espera recibir la que tendrá que incluir en el nuevo INVITE para conseguir la transparencia. Incluye además en la invocación un identificador único para la sesión, que será el call-id del INVITE inicial. Los pasos que realiza el Transparency SBB en la ejecución del método son:
  - Guarda en un campo CMP el identificador de sesión, para poder acceder a él en las siguientes invocaciones que reciba. Al ser la misma *SBB Entity* la que gestiona toda la transparencia de la sesión, siempre tendrá este dato.
  - Recorre el contenido de la carga SDP recibida, y acorde a él construye el objeto `SessionInfo` necesario para interactuar por RMI con el Transparency Manager.
  - Invoca por RMI al Transparency Manager para que le devuelva la información de las direcciones IP y puertos a utilizar. Utiliza el método `addEarlySession`.
  - Conforme a lo recibido al invocar al Transparency Manager, construye la carga SDP a devolver al B2BUA SBB.

El resultado de invocar a este método es una carga SDP similar a la que se recibe, pero en la que cambian las direcciones IP y los puertos, que serán los que se utilizarán en el Transparency Manager.

La Tabla 4 muestra un ejemplo de las cargas SDP correspondientes a los INVITE de ejemplo de la Tabla 1, en la que se aprecian los cambios comentados. El Transparency Manager se encuentra también en la IP 192.168.0.192.

v=0 o=isma 3552730820 3552730820 IN IP4 192.168.0.192 s=A Funky MONSTER Stream t=0 0 m=audio **23010** RTP/AVP 0 8 14 101 c=IN IP4 **192.168.0.192** a=rtpmap:0 PCMU/8000 a=rtpmap:8 PCMA/8000 a=rtpmap:14 MPA/8000 a=rtpmap:101 telephone-event/8000 m=video **23012** RTP/AVP 34 96 32 97 c=IN IP4 **192.168.0.192** a=rtpmap:34 H263/90000 a=rtpmap:96 H263-1998/90000 a=rtpmap:32 MPV/90000 a=rtpmap:97 MP4V-ES/90000	v=0 o=isma 3552730820 3552730820 IN IP4 192.168.0.192 s=A Funky MONSTER Stream t=0 0 m=audio **50054** RTP/AVP 0 8 14 101 c=IN IP4 **192.168.0.192** a=rtpmap:0 PCMU/8000 a=rtpmap:8 PCMA/8000 a=rtpmap:14 MPA/8000 a=rtpmap:101 telephone-event/8000 m=video **50006** RTP/AVP 34 96 32 97 c=IN IP4 **192.168.0.192** a=rtpmap:34 H263/90000 a=rtpmap:96 H263-1998/90000 a=rtpmap:32 MPV/90000 a=rtpmap:97 MP4V-ES/90000

Tabla 4: Carga SDP INVITE

- `getConfirmedSessionSDP(byte[] sdpBytes)`: Una vez recibida la respuesta 200 OK que indica un establecimiento correcto de sesión, el B2BUA SBB invoca este método del Transparency SBB pasándole la carga SDP recibida del llamado, y esperando recibir la carga a incluir en la respuesta a enviar al llamante.

  Los pasos a realizar son similares a los del método anterior, con la diferencia de que el método a invocar del Transparency Manager en este caso es `addConfirmedSession`, y que el identificador de la sesión lo obtiene del campo CMP.

  Los cambios en el SDP son también similares a los que se observan en la Tabla 4, y aparecen en la Tabla 5

v=0 o=carmen 3552730820 3552730820 IN 192.168.0.192 IP4 s=A Funky MONSTER Stream t=0 0 m=audio **50004** RTP/AVP 0 101 c=IN IP4 **192.168.0.192** a=rtpmap:101 telephone-event/8000 m=video **50064** RTP/AVP 34 c=IN IP4 **192.168.0.192**	v=0 o=carmen 3552730820 3552730820 IN 192.168.0.192 IP4 s=A Funky MONSTER Stream t=0 0 m=audio **23006** RTP/AVP 0 101 c=IN IP4 **192.168.0.192** a=rtpmap:101 telephone-event/8000 m=video **23008** RTP/AVP 34 c=IN IP4 **192.168.0.192**

Tabla 5: Carga SDP 200 OK

- `removeEarlySession()`: Si se produce un fallo en el establecimiento de la sesión, por cualquiera de las dos causas vistas en 4.1.2, se invoca este método del Transparency SBB.

  Lo que se hace en este método es obtener el identificador de la sesión y directamente invocar el método `removeEarlySession` del Transparency Manager.

- `removeConfirmedSession()`: Se invoca este método una vez se libera de forma correcta la sesión, tal y como se ve en 4.1.3. En este caso el comportamiento es invocar directamente el método `removeConfirmedSession` del Transparency Manager.

Como se puede observar, estos métodos síncronos lo que hacen es encapsular las llamadas al Transparency Manager, así como el código necesario para preparar estas llamadas y procesar las respuestas que ofrecen. Si cambiase la manera de interactuar con el Transparency Manager, se cambiaría la implementación de estos métodos síncronos, pero sin ser necesario cambiar su firma, la forma de invocarlos.

Existen además otros métodos que permiten gestionar la transparencia si se producen transferencias de sesiones. Se verán en el siguiente capítulo al analizar los SBBs encargados de la gestión de dichas transferencias.

Para finalizar con este apartado, se denota el hecho de que la versión actual del Mobility Manager únicamente soporta la negociación de la sesión tal y como se ha visto, mediante el intercambio de SDPs entre la petición INVITE (que lleva la oferta) y la respuesta 200 OK (que lleva la repuesta) [7]. Existen otras formas de negociar la sesión, como puede ser que la oferta vaya en el 200 OK y la respuesta en el ACK, no llevando carga SDP el INVITE inicial, usando respuestas provisionales, la petición UPDATE, etc. Del mismo modo tampoco se contempla actualmente el hecho de poder renegociar el estado de una sesión.

Para poder tratar otras formas de negociar la sesión o soportar renegociaciones, sería necesario extender el funcionamiento básico ya explicado del B2BUA SBB. Esto sería una posible mejora al proyecto, tal y como se refleja en Capítulo 8.

Capítulo 5

Mobility Manager. Gestión de las transferencias de sesión

Este capítulo complementa al anterior, finalizando el análisis de cómo funciona y se ha implementado el Mobility Manager. Lo que queda por ver es cómo se realizan las transferencias de sesión, y para ello en este capítulo se analizarán los componentes encargados de que puedan llevarse a cabo, Pull Transfer SBB y Push Transfer SBB, y también los cambios necesarios en los componentes ya explicados en el capítulo anterior.

El punto de partida consiste en que se tiene establecida de forma correcta una sesión entre dos extremos, tal y como se ha visto en 4.1.1, y que todo el flujo de datos que intercambian pasa a través del Transparency Manager. Uno cualquiera de los extremos, que tiene activo el servicio de movilidad, decide que quiere transferir la sesión a otro de sus terminales, y para ello tiene dos opciones:

- Desde su otro terminal solicita traerse la sesión que tiene activa, lo que se conoce como transferencia de tipo *Pull*, y para ello manda un INVITE indicando los datos de la sesión que quiere obtener.
- Desde el terminal en que tiene establecida la sesión indica que quiere llevársela a otro, lo que se conoce como transferencia de tipo *Push*, mandando para ello un REFER indicando la dirección del nuevo terminal.

El Pull Transfer SBB será el encargado de gestionar las transferencias conocidas como Pull Transfer, mientras que las Push Transfer las gestionará el Push Transfer SBB. A continuación pasa a detallarse el funcionamiento de ambos SBBs.

5.1 Pull Transfer SBB

Las transferencias de tipo *Pull* conllevan una menor señalización, y por ello fueron las primeras por las que se comenzó la implementación. El Pull Transfer SBB será un SBB hijo del B2BUA SBB, y serán necesarios una serie de cambios dentro de este último para que pueda invocar de forma correcta a su SBB hijo, y se pueda llevar a cabo la transferencia.

A la hora de analizar el funcionamiento se va a proceder por pasos, que además seguirán el flujo de la implementación. Lo primero será ver cómo se consigue que se invoque la funcionalidad y que se empiece a procesar la transferencia. Posteriormente se analizará el funcionamiento del SBB desde el punto de vista de cómo se maneja la señalización SIP, para que el flujo de mensajes SIP quede implementado de forma correcta. Lo último será ver cómo se interacciona con el Transparency Manager para que se consiga la transparencia de los datos.

5.1.1 Inicio del proceso de transferencia

Una transferencia de tipo *Pull* comienza con el usuario mandando un INVITE desde el terminal al que quiere que se le transfiera la sesión. La peculiaridad de este INVITE es que contiene una cabecera Replaces en la que se indica cuál es la sesión que quiere que le sea transferida (indicando el call-id y los tags que identifican al diálogo SIP actualmente establecido), pero por lo demás es un INVITE con un nuevo call-id, y que iniciará un diálogo totalmente distinto al que ya tiene establecido el usuario en el otro terminal.

La Figura 12 muestra un ejemplo de este INVITE, que serviría para transferir la sesión establecida con los mensajes de Tabla 1 y Tabla 2. Las condiciones son las mismas que las comentadas para dichas tablas, y ahora el usuario con movilidad manda el INVITE desde un nuevo terminal donde tiene registrada la identidad sip:isma.movil@open-ims.test. La IP donde está este terminal es la misma que la del antiguo, 192.168.0.192. Lo más importante es la cabecera Replaces, que aparece marcada en negrita.

```
INVITE sip:carmen@open-ims.test SIP/2.0
Via: SIP/2.0/UDP 192.168.0.192:5063;branch=z9hG4bK-8837-1-0
From: isma.movil <sip:isma.movil@open-ims.test>;tag=1
To: carmen <sip:carmen@open-ims.test>
Call-ID: 1-8837@192.168.0.192
CSeq: 1 INVITE
Contact: <sip:isma.movil@192.168.0.192:5063;transport=UDP>
Replaces: f3b3df97e6e9d0ff260c68053da9342d@192.168.0.192;to-tag=
1729a286;from-tag=1006
Max-Forwards: 70
Subject: IMS Call
Route: <sip:orig@scscf.open-ims.test:6060;lr>
Content-Type: application/sdp
Content-Length:   235

v=0
o=isma.movil 5365576588371 2353687637 IN IP4 192.168.0.192
s=-
c=IN IP4 192.168.0.192
t=0 0
m=audio 6000 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
m=video 6002 RTP/AVP 34
a=rtpmap:34 H263/90000
```

Figura 12: INVITE para transferencia de tipo *Pull*

El objetivo que se persigue es que el B2BUA SBB reciba el INVITE, y que cree un SBB hijo de tipo Pull Transfer SBB que se encargue de procesar la transferencia, una nueva *SBB Entity*. Si la transferencia se realiza de forma correcta el Pull Transfer SBB habrá creado un nuevo diálogo entre el nuevo terminal del usuario y el Mobility Manager, finalizando el existente con el antiguo terminal. Para terminar su ejecución deberá indicar al B2BUA SBB que este diálogo sustituye al anterior que había, de tal manera que este último SBB pueda gestionar posibles nuevas transferencias, o directamente la liberación de la sesión. Lo que se hace si la

transferencia es correcta es cambiar el extremo de una de las dos patas del B2BUA, aquella en la que se produce la transferencia.

El INVITE se enruta a través del Core IMS, y para que llegue al Mobility Manager lo único necesario es que se origine desde un usuario que tenga el servicio de movilidad activo, para lo cual todo debe estar configurado como se ha visto en 3.1.

El INVITE debe ser procesado por la misma *SBB Entity* del B2BUA SBB que esté tratando la sesión ya establecida sobre la que se desea hacer una transferencia, y para conseguir esto es necesario un cambio en la forma en que se procesan los eventos INVITE. Actualmente cada INVITE recibido lanza una nueva ejecución del servicio, creando para ello una nueva *SBB Entity* tal y como se ha visto en 4.1. Ahora hay que tener en cuenta si el INVITE busca hacer una transferencia de tipo *Pull*, si tiene una cabecera Replaces que haga referencia a un diálogo ya establecido. Lo que se hace es procesar previamente cada INVITE en un método selector de evento inicial [16], cuyo comportamiento será el siguiente:

- Si el INVITE no contiene la cabecera Replaces significará que será un evento inicial, y se lanzará una nueva ejecución del servicio creando una nueva *SBB Entity*. Se computará un nombre de convergencia propio, que será el call-id del INVITE.
- En caso de que la contenga, se obtendrán los valores de call-id y tags del diálogo al que se pretende sustituir. Se buscará si este diálogo forma parte de alguna de las sesiones que se estén tratando (posteriormente se verá cómo se puede abordar esto), dándose los siguientes casos:
    - Si hace referencia a alguna sesión, se buscará el identificador de ésta, y se utilizará como nombre de convergencia para el evento. De esta forma, el evento llegará a la misma *SBB Entity* que esté tratando la sesión, pues el identificador de sesión será el mismo que se usó cuando se creó inicialmente.
    - Si no hace referencia a una de las sesiones activas, se marca el evento para que sea procesado por una *SBB Entity* totalmente nueva.

Una vez elegida la *SBB Entity* a la que se entregará el INVITE, ésta lo procesará, y lo primero que hará será comprobar nuevamente si el INVITE contiene la cabecera Replaces. Así sabrá si lo que tiene que procesar es un nuevo establecimiento de sesión, o un intento de transferencia de tipo *Pull*.

En caso de ser un nuevo establecimiento de sesión se procederá como se ha visto en 4.1, pero es necesario añadir la lógica que permita que se lleve a cabo el procedimiento en el método selector de evento inicial visto anteriormente. Es necesario tener almacenada la información de todas las sesiones activas, consistiendo esta información en los dos diálogos que la componen, las dos patas del B2BUA; y esta información debe ser accesible desde el método selector, que se ejecuta en un *SBB Object* no asignado a ninguna *SBB Entity*.

La solución escogida consiste en tener la información guardada en un mapa, cuya clave será el call-id del INVITE inicial que arranca el servicio, que es el valor que se usa en el método selector para dirigir a la *SBB Entity* correspondiente. Los valores del mapa serán unos objetos que guardarán la información de los dos diálogos que conformarán la sesión, los conocidos como diálogo entrante y diálogo saliente del B2BUA. Todos estos objetos estarán recogidos en

la librería del servicio. A su vez, el mapa será un campo estático de la clase que representa al SBB, de tal forma que su información pueda ser accedida desde cualquier *SBB Object*, represente éste o no a una *SBB Entity* concreta.

Al recibir el INVITE inicial del servicio, se crea una nueva entrada en el mapa, añadiendo la información de los diálogos. Del diálogo entrante la información es completa, del saliente faltaría el tag remoto, que se añadiría si se establece de forma correcta la sesión al recibir la respuesta 200 OK. Cuando la sesión se libera, o si falla el establecimiento de sesión, se elimina la información de la tabla. Del mismo modo, si se produce una transferencia de forma correcta, la información debe actualizarse, tal y como se verá posteriormente.

Todo lo anterior se centraba en que el INVITE no tuviera la cabecera Replaces, e hiciese referencia a un nuevo establecimiento de sesión. En caso de que sí que contenga la cabecera, el procedimiento a seguir será el siguiente:

- Si el INVITE llega a una *SBB Entity* nueva, lo cual significa que la cabecera Replaces no hace referencia a ningún diálogo de las sesiones actualmente activas, se responde con un mensaje 481 Call/Transaction Does Not Exist [6].
- Si llega a una SBB Entity que está controlando una sesión, se busca en cuál de los diálogos se está intentando la transferencia, y se crea un nuevo SBB hijo de tipo Pull Transfer SBB, una nueva *SBB Entity*, que será quien la procese. Se le invoca primero de forma síncrona pasándole los datos necesarios para el procesamiento de la transferencia, y posteriormente se suscribe el SBB al Activity Context Interface por donde se ha recibido el INVITE, para que éste también pueda recibirlo y realizar la lógica de la transferencia.

Por cada intento de transferencia se creará una nueva *SBB Entity* del Pull Transfer SBB que será la encargada de procesarla.

5.1.2 Manejo de la transferencia

El Pull Transfer SBB necesita cierta información para poder llevar a cabo la transferencia. Una buena forma de obtener esta información podría ser que al recibir el INVITE que inicia el proceso de transferencia, el B2BUA SBB guardase la información necesaria en el *Activity Context Interface* (ACI), y el Pull Transfer SBB accediese a ella al recibir el evento.

El problema es que entre la información que se necesita aparecen referencias al Transparency SBB encargado de manejar la transparencia de la sesión, así como al propio B2BUA SBB, pues tiene que ser notificado si la transferencia se completa para actualizar la información de los diálogos que procesa. Estas referencias a otros SBBs, que son referencias a *SBB Entity*, no pueden ser utilizadas como variables compartidas en un ACI [16], por lo que se emplea la llamada síncrona descrita en el subapartado anterior.

Se define por tanto una *SBB Local Interface* para el Pull Transfer SBB con un método síncrono. Al invocar este método síncrono se pasan todas las variables necesarias para manejar la

transferencia, que se guardarán en campos CMP. Así al recibir el INVITE el SBB ya tiene toda la información necesaria. Esta información consistirá en las referencias a los dos SBBs ya descritas anteriormente, un booleano indicando si el intento de transferencia se está produciendo en el diálogo entrante o en el saliente, el propio diálogo que está siendo transferido, y finalmente la repuesta 200 OK que sirvió para establecer de manera correcta la sesión.

El procesamiento comienza cuando se recibe el INVITE, y llevarán a cabo los siguientes pasos para la realización de la transferencia:

- Se crea un nuevo diálogo a partir del INVITE, así como un ACI al que el SBB se suscribe. Aquí se podría comprobar de alguna forma si quien está pidiendo que se le transfiera la sesión está autorizado a hacerlo, permitiendo la transferencia sólo en caso de que así sea. Pero esto no se va a hacer, se supone que mediante algún mecanismo externo sólo el mismo usuario es capaz de conocer qué sesiones activas tiene, y de formar correctamente el INVITE con la cabecera Replaces. Dado esto, no sería necesaria ninguna comprobación adicional en el Mobility Manager. Esto se comenta en el Capítulo 8.
- A partir de la respuesta 200 OK que se tiene almacenada en campo CMP por un lado, y del diálogo que se acaba de crear por otro, se construye una respuesta 200 OK para contestar al INVITE y aceptar la transferencia. Hay además una interacción con el Transparency SBB para obtener el SDP correcto a incluir en la respuesta, que se analizará en el próximo subapartado.

Se puede observar que el INVITE lo procesa y maneja directamente el Mobility Manager, el extremo de la comunicación que no está realizando la transferencia en ningún momento tiene constancia de que ésta se esté produciendo. El Mobility Manager se comporta como el destino lógico para el usuario que inicia la transferencia, y su dirección aparecerá en la cabecera Contact del mensaje 200 OK enviado, tal y como se vio en 4.1.1.

Lo siguiente será recibir un ACK que confirme el diálogo. Cuando esto suceda los pasos a realizar serán los que se enumeren a continuación, y que conducirán a que la transferencia se complete satisfactoriamente:

- Se invoca al Transparency Manager para que lleve a cabo la transparencia de los datos, se analizará más detenidamente en el siguiente subapartado.
- Se invoca de forma síncrona al SBB padre, el B2BUA SBB, indicándole el nuevo diálogo y en cuál de los dos extremos del B2BUA se ha producido la transferencia (información que ya le había indicado el propio padre en el inicio del procesamiento). Este es un procedimiento que hay que implementar en el B2BUA SBB, y lo que se hace es:
    - Según el extremo en que se haya producido la transferencia, actualiza el mapa de las sesiones sustituyendo la información del diálogo correspondiente. Esto permite que se puedan producir nuevas transferencias de tipo *Pull*, siguiendo todo el procedimiento que ha sido detallado.
    - En el ACI que representa al nuevo diálogo, indicar como diálogo asociado el que se tiene con el extremo que no se ha movido. De manera simétrica, actualizar el diálogo asociado en el ACI del diálogo con el extremo que no se ha

movido, para que éste haga referencia al nuevo diálogo. De esta forma quedan actualizadas las referencias en las dos patas del B2BUA, lo que permite que pueda producirse de forma correcta la liberación de la sesión tal y como se detalla en 4.1.3, y también que sea posible efectuar transferencias de tipo *Push*, tal y como se verá en 5.2.

- El Pull Transfer SBB se desuscribe del nuevo diálogo y suscribe a su SBB padre. Ya se ha realizado la transferencia, se ha sustituido el diálogo de una de las patas del B2BUA y el resto de mensajes los procesará directamente el B2BUA SBB. Al hacer esto el B2BUA SBB recibirá el ACK, pero éste no realiza ninguna acción al recibir el evento correspondiente, como se vio en 4.1.1.
- Se termina el diálogo que ha sido transferido. Para ello se construye y envía un BYE acorde al diálogo, con sus identificadores y rutas, tal y como se hace en 4.1.3. El Pull Transfer SBB se suscribe al diálogo mientras que desuscribe a su SBB padre; de esta forma, la respuesta 200 OK que confirme que el diálogo ha terminado la recibirá únicamente el Pull Transfer SBB.

Lo último será recibir la respuesta 200 OK mencionada en el último punto, que confirma el final del diálogo que ha sido transferido. No es necesario realizar ninguna acción al recibir esta respuesta.

5.1.3 Interacción con el Transparency SBB

Como se ha visto en el subapartado anterior, hay dos momentos en que el Pull Transfer SBB utiliza el Transparency SBB para manejar la transparencia de los datos de la sesión. El objetivo es que todos los datos que esté enviando el extremo que no se mueve se redirijan de forma correcta al nuevo terminal del usuario al que se ha transferido la sesión, y que de forma simétrica los datos que se envíen desde este nuevo terminal lleguen al extremo que no se mueve.

La primera interacción se produce cuando el Pull Transfer SBB debe contestar con un 200 OK al INVITE que inicia el proceso de transferencia. En ese momento, invoca de forma síncrona al Transparency SBB para que éste le devuelva el SDP a incluir en el cuerpo de la respuesta, y para ello le indica en qué dialogo se ha producido la transferencia, y también el SDP recibido en el INVITE. Es necesario ampliar la funcionalidad del Transparency SBB descrita en 4.2 para poder llevar a cabo esta operación, se detallan a continuación los cambios:

- En el método `getEarlySessionSDP` se guarda en un campo CMP la información de las direcciones IP y puertos que devuelve el Transparency Manager. Estas direcciones IP y puertos serán los que el Transparency Manager utilice en el diálogo saliente.
- En el método `getConfirmedSessionSDP` se guarda en un campo CMP el SDP que se devuelve. Este SDP contiene la información de las direcciones IP y puertos que el Transparency Manager utiliza en el diálogo entrante; además, y teniendo en cuenta lo que se ha visto en 4.2.2 acerca de cómo se ha supuesto que es la negociación de la sesión, el SDP contiene los códecs ya negociados y que están siendo utilizados.

Volviendo al método síncrono, la forma de construir el SDP a devolver dependerá de en cuál de los dos diálogos se esté produciendo la transferencia:

- Si es en el diálogo entrante, se construye directamente a partir del SDP de dicho diálogo guardado en un campo CMP. Contiene tanto los códecs negociados, como las direcciones IP y puertos usados en el Transparency Manager.
- Si es en el saliente se construye también a partir del SDP del diálogo entrante, pues contiene los códecs negociados, pero actualizando las direcciones IP y puertos a aquellas que el Transparency Manager está usando en el diálogo saliente, y que se obtienen del campo CMP.

Para lo único que se utiliza el SDP recibido en el INVITE es para actualizar los valores de la línea Origin del SDP a devolver que hacen referencia a los identificadores de sesión, de tal forma que sean iguales a los recibidos. La Figura 13 muestra el SDP que se recibiría acorde al INVITE enviado que se muestra en la Figura 12.

```
v=0
o=carmen 5365576588371 2353687637 IN 192.168.0.192 IP4
s=A Funky MONSTER Stream
t=0 0
m=audio 50004 RTP/AVP 0 101
c=IN IP4 192.168.0.192
a=rtpmap:101 telephone-event/8000
m=video 50064 RTP/AVP 34
c=IN IP4 192.168.0.192
```

Figura 13: Carga SDP del 200 OK en transferencia de tipo *Pull*

Con este comportamiento se está realizando una asunción, que el INVITE recibido contendrá una oferta de códecs compatible con los que se están usando en la sesión. Al igual que se ha supuesto que el usuario, mediante algún procedimiento externo, es capaz de conocer qué sesiones tiene establecidas para poder hacer una transferencia de tipo Pull, se va a suponer ahora que también es capaz de conocer los códecs que se están utilizando en dichas sesiones. En consecuencia, el cliente es capaz de mandar un INVITE tanto con una cabecera Replaces correcta, como con un contenido SDP compatible. Si el contenido no fuese compatible, el cliente lo que haría al recibir la respuesta sería mandar un ACK y posteriormente un BYE para liberar la sesión.

La segunda y última interacción con el Transparency SBB se produce en el procesamiento del ACK, una vez que se considera completa la transferencia de la sesión. En ese momento lo que se quiere es interactuar directamente con el Transparency Manager para indicarle que se ha producido una transferencia y que tiene que actuar para que se mantenga la transparencia de los datos. Desde el Transparency SBB se invocará el método que quedó por analizar en 4.2.1.

La firma del método es `transferSession(String id, SessionInfo userAgentSessionInfo, boolean incomingTransfer)`. En él se pasan como parámetros el id único que identifica la sesión, un booleano indicando en cuál de los diálogos se ha producido la transferencia, y un objeto `SessionInfo` con la información de las direcciones IP y puertos que usará el nuevo terminal, y que pasará a ser el nuevo extremo de la sesión.

El método síncrono del Transparency SBB será el que encapsule esta llamada, al igual que se vio en los métodos de 4.2.2. La firma del método es `transferSession(byte[] sdpBytes, boolean incomingTransfer)`. El Pull Transfer SBB invoca este método pasándole la información SDP recibida en el INVITE, así como el booleano indicando donde se produce la transferencia. El Transparency SBB obtiene el identificador de sesión del campo CMP, construye el objeto `SessionInfo` a partir del SDP e invoca por RMI al Transparency Manager.

5.2 Push Transfer SBB

Las transferencias de tipo *Push* conllevan una mayor señalización SIP. Aún así, y tal y como se irá viendo en el desarrollo del apartado, su implementación será algo más sencilla y no serán necesarias tantas asunciones desde el punto de vista del usuario. Esto se debe a que ahora la transferencia se inicia desde el terminal que tiene establecida la sesión, y el usuario lo único que tendrá que indicar es la dirección con que esté registrado el nuevo terminal, y lo hará con un mensaje que no será un INVITE y que no tendrá contenido SDP.

La forma de analizar la implementación será similar a como se hizo en el caso anterior. En primer lugar se verá como es el inicio del proceso de transferencia, posteriormente se analizará cómo se lleva a cabo desde el punto de vista de la señalización, y finalmente se explicará la interacción para conseguir la transparencia de los datos.

5.2.1 Inicio del proceso de transferencia

En este caso el proceso de transferencia se inicia mediante el envío de una petición REFER por parte del usuario que tiene establecida la sesión. Con el envío de este mensaje indicará que desea transferir la sesión que tiene establecida en ese momento a otro de sus terminales, y cuya dirección vendrá indicada en la cabecera Refer-To del mensaje.

Este REFER irá dirigido hacia el Mobility Manager, debido a que es el extremo lógico con el que se mantiene la comunicación al estar actuando en modo B2BUA. Este hecho será el que permita que se pueda realizar la transferencia de forma transparente, sin la intervención del otro extremo. Toda la señalización afectará únicamente a una de las patas del B2BUA, cuyo extremo cambiará en caso de que la transferencia se produzca de forma correcta.

Lo más importante en este tipo de transferencia es que el REFER se puede enviar dentro del mismo diálogo SIP que está ya establecido, o en un diálogo aparte que haga referencia al primero [9]. La primera de las formas es la única que en un primer momento se decide implementar, por las siguientes razones:

- Al ir en el mismo diálogo que el INVITE seguirá su mismo camino y llegará seguro a su destino, el Mobility Manager, sin necesidad de realizar ningún cambio. Si fuese en otro

diálogo habría que añadir una nueva regla al core IMS para que los mensajes REFER se redirigieran al Mobility Manager.

- Es más sencilla de tratar, el diálogo que se quiere transferir es sobre el que viene la petición, por lo que la asociación es inmediata. De la otra manera habría que hacer un procedimiento similar al que se hizo para el caso de las transferencias de tipo Pull en 5.2.1, analizando la cabecera Target-Dialog en que se indicaría el diálogo que se quiere transferir.
- Es la forma de proceder que tiene myMonster, el softphone IMS que se utiliza para probar el servicio. La otra forma únicamente podría probarse con scripts SIPp, al igual que ocurre con las transferencias de tipo Pull.

La Figura 14 muestra un ejemplo de cómo sería el mensaje REFER a enviar para iniciar el proceso de transferencia. El caso sería similar al comentado en 5.1.1 para las transferencias de tipo *Pull*, desde el terminal con el que el usuario con movilidad tiene establecida una sesión, según los mensajes de la Tabla 1, manda un REFER para transferir la sesión al nuevo terminal registrado con la identidad sip:isma.movil@open-ims.test.

```
REFER sip:192.168.0.192:5060;transport=udp SIP/2.0
Via: SIP/2.0/UDP
192.168.0.192:5081;branch=z9hG4bK641ff2bdab10b975661723b24b6f7b4c383635
To: <sip:carmen@open-ims.test>;tag= 1729a286
Call-ID: f3b3df97e6e9d0ff260c68053da9342d@192.168.0.192
From: <sip:isma@open-ims.test>;tag= 1006
Contact: <sip:192.168.0.192:5081;transport=udp>
Route: <sip:mo@pcscf.open-ims.test:4060;lr>,<sip:mo@scscf.open-
ims.test:6060;lr>
Max-Forwards: 20
CSeq: 10 REFER
Refer-To: <sip:isma.movil@open-ims.test;method=INVITE>
User-Agent: Fokus MONSTER Version: 0.9.13
Content-Length: 0
```

Figura 14: REFER para transferencia de tipo *Push*

Para que el evento que representa al REFER llegue a la misma SBB Entity del B2BUA SBB que está tratando la sesión, lo único que hay que hacer es declarar en el SBB que se quiere recibir dicho evento. Este evento le será entregado directamente al SBB al producirse en uno de los dos diálogos que está manejando y a los que se encuentra suscrito.

Una vez se reciba el evento, lo que se hará será crear un SBB hijo de tipo Pull Transfer SBB al que invocará de forma síncrona para facilitarle la información necesaria para que procese la transferencia, y al que suscribirá al diálogo en que se ha recibido el evento para que lo procese. El procedimiento es similar al visto en 5.1.1, compartiendo el mismo objetivo: que el Push Transfer SBB sea el encargado de manejar la transferencia, y que en caso de que ésta se produzca de forma correcta notifique al B2BUA SBB para que sustituya uno de sus diálogos.

Dado que la señalización SIP correspondiente a la transferencia irá sobre el mismo diálogo que se pretende sustituir, el B2BUA SBB se desuscribe temporalmente del diálogo, para que los eventos relativos a los mensajes de señalización propios de la transferencia los procese únicamente el Push Transfer SBB.

5.2.2 Manejo de la transferencia

Se realiza una invocación síncrona al Push Transfer SBB por los mismos motivos expuestos en 5.1.2, la necesidad de pasar referencias a otros SBBs. La información que se facilita en este caso la componen las referencias al SBB padre y al Transparency SBB, un booleano indicando en qué diálogo se está intentando hacer la transferencia, y el INVITE inicial que lanzó la ejecución del servicio. Todos estos datos se guardan en campos CMP.

Al recibir el REFER se contesta inmediatamente con una respuesta 202 Accepted, que implica que se aceptan el REFER y la suscripción que éste conlleva [10][11], y por la que se tiene que informar al extremo que ha enviado el REFER de cómo evoluciona su petición. La primera información es una petición NOTIFY con contenido 100 Trying, que indica que se está intentando realizar la transferencia. El estado de la suscripción en el NOTIFY se marca como activo, y se indica un tiempo de expiración de la suscripción que debe ser superior al tiempo permitido para realizarla, y sobre el que se volverá posteriormente.

La última acción a realizar en el procesamiento del REFER consiste en construir y enviar el INVITE que irá dirigido al nuevo terminal al que el usuario quiere transferir la sesión. Los pasos para conseguirlo son:

- Se obtiene la dirección del nuevo terminal de la cabecera Refer-To. Esta dirección se puede denominar como objetivo de la transferencia.
- Se crea un nuevo diálogo SIP. Para ello se toma como destino el objetivo de la transferencia, y como origen el otro extremo de la sesión a transferir, el que no se mueve.
- A partir de este diálogo, y tomando como referencia el INVITE inicial del servicio que se obtuvo gracias al SBB padre, se construye el nuevo INVITE a ser enviado para poder transferir la sesión.
- Se interactúa con el Transparency SBB para obtener el SDP a incluir en el INVITE que permita la transparencia de la sesión. Se analizará con más detalle en el próximo subapartado.
- Se crea un Activity Context Interface que represente al diálogo, y el SBB se suscribe a él para poder procesar el resto de mensajes pertenecientes a dicho diálogo.
- Se envía el INVITE.

Son necesarias un par más de consideraciones que implican algo más de procesamiento antes de que se envíe el INVITE. Estas consideraciones tienen que ver en primer lugar con la forma en que se tiene que enrutar el INVITE a través del Core IMS, para permitir que éste se procese de forma correcta y llegue a su destino; y en segundo lugar con el comportamiento que se espera en la transferencia, y que implicará que se fije un tiempo para que el destino responda.

Respecto al tema de enrutar a través del Core IMS, ahora no se está en el mismo caso que en 4.1.1, cuando el INVITE pese a pertenecer a un nuevo diálogo tenía relación con otro, y así lo reconocía el Core IMS. Ahora el Application Server crea un INVITE totalmente independiente de cara al Core IMS, actuará como un Initiating User Agent [2]. Aquí existen dos opciones: que genere el INVITE en nombre de un usuario, a través de su identidad IMPU, o bien que lo

genere en nombre de un PSI que represente al servicio[1]. La diferencia vendrá según lo que aparezca en la cabecera P-Asserted-Identity.

La primera opción implicaría suplantar al usuario que no está realizando la transferencia, y además que se evaluasen los posibles servicios que el usuario tenga activos cuando origina una llamada. El Mobility Manager no debería suplantar a dicho usuario, sobre todo si además este usuario no tiene activo el servicio de movilidad. Se opta por la segunda opción, definiendo en el Core IMS un PSI para el servicio, y enrutando el INVITE a través del S-CSCF que lo sirve. No se define ningún tratamiento para los INVITEs que genere el PSI, de tal forma que el S-CSCF los enrutaría directamente al S-CSCF que sirve al destino, al objetivo de la transferencia. Estos aspectos ya se comentaron en 3.1.

La Figura 15muestra un ejemplo del INVITE que se enviaría, relacionado con el REFER de la Figura 14, y todo el resto de particularidades anteriores.

```
INVITE sip:isma.movil@open-ims.test SIP/2.0
CSeq: 17 INVITE
User-Agent: Fokus MONSTER Version: 0.9.13
Via: SIP/2.0/UDP 192.168.0.192:5060;branch=z9hG4bK-353031-
37648d937cfcb682fccf21d2ddc4d85a
Max-Forwards: 70
Call-ID: 5f4c962b6875dcfa4c1eb471b90a47e8@192.168.0.192
Contact: <sip:192.168.0.192:5060;transport=udp>
From: <sip:carmen@open-ims.test>;tag=9f775088
To: <sip:isma.movil@open-ims.test;method=INVITE>
Route: <sip:orig@scscf.open-ims.test:6060;lr>
P-Asserted-Identity: <sip:mobility.manager@open-ims.test>
Content-Type: application/sdp
Content-Length: 229

v=0
o=carmen 3552730820 3552730820 IN 192.168.0.192 IP4
s=A Funky MONSTER Stream
t=0 0
m=audio 50004 RTP/AVP 0 101
c=IN IP4 192.168.0.192
a=rtpmap:101 telephone-event/8000
m=video 50064 RTP/AVP 34
c=IN IP4 192.168.0.192
```

Figura 15: INVITE para transferencia de tipo *Push*

Yendo ahora a la segunda de las consideraciones, una vez el INVITE llegue al objetivo de la transferencia se van a considerar tres posibles escenarios, similares a los casos del establecimiento básico de sesión expuestos en 4.1, y que representarán los tres comportamientos posibles en las transferencias de sesión de tipo *Push*.

- El objetivo de la transferencia contesta al Invite con un 200 OK., con lo que la transferencia se completa.
- El objetivo de la transferencia rechaza el INVITE con una respuesta de error, lo que finalizaría el intento de transferencia. Lo más habitual es que rechace con una respuesta 486 Busy Here.

- Que tras un tiempo prudencial el objetivo de la transferencia no conteste, en cuyo caso el Mobility Manager decidirá cancelar el INVITE, finalizando el intento de transferencia.

Se estudiará a continuación cómo se ha implementado cada uno de los tres posibles escenarios.

5.2.2.1 Transferencia realizada de forma correcta

Este escenario se da cuando se recibe una respuesta 200 OK para el INVITE. Los pasos a realizar en la recepción de la respuesta, y que terminarán con la transferencia de sesión completada de forma correcta son:

- Se envía inmediatamente un ACK para confirmar el diálogo que implica la transferencia.
- A través del diálogo que está siendo transferido se envía un NOTIFY con cuerpo 200 OK, el cual indica que se ha terminado de forma correcta la petición del REFER, y al mismo tiempo implica que se termine la suscripción establecida por dicho REFER.
- Se invoca al Transparency SBB para indicar que se ha producido la transferencia y que mantenga la transparencia de los datos, se verá en más detalle en 5.2.3.
- Se manda un Bye por el diálogo que está siendo transferido, ya está hecha la transferencia y se termina el diálogo con el terminal del usuario que ha iniciado la transferencia.
- Se invoca al SBB padre de forma síncrona, para que actualice el diálogo de la pata del B2BUA donde se ha producido la transferencia, de la misma manera que se hace en las transferencias de tipo *Pull* y que se explicó en 5.1.2.
- El SBB se desuscribe del nuevo diálogo, y suscribe a su SBB padre. De esta forma el Push Transfer SBB acaba su ejecución habiendo completado la transferencia, y el B2BUA SBB vuelve a tener una sesión establecida entre dos extremos, y queda a la espera de otros posibles intentos de transferencia o de que la sesión se libere.

El B2BUA SBB recibirá la respuesta 200 OK al haber sido suscrito al diálogo. Para que no procese esta respuesta como si se tratase de un establecimiento correcto de sesión, el Push Transfer SBB indica en un booleano en el ACI que la respuesta ya ha sido procesada. Cuando el B2BUA SBB recibe una repuesta 200 OK de INVITE, lo primero que hace es consultar si en el ACI hay un booleano indicando que ya ha sido tratada, y que indicaría que es un 200 OK proveniente de una transferencia correcta de sesión de tipo *Push*; si es así no la procesa, y si no continuaría con la ejecución vista en 4.1.1.

Se reciben en el SBB eventos correspondientes a respuestas 200 OK para el NOTIFY y para el BYE enviados por el diálogo que está siendo transferido. No es necesario procesar estas respuestas.

5.2.2.2 Rechazo por parte del objetivo de la transferencia

El escenario se produce cuando se recibe una respuesta de error en el cliente para el INVITE. Implica que el intento de transferencia ha fallado, y el diálogo SIP que se mantiene con el objetivo de la transferencia finaliza. Las acciones a realizar son:

- Mandar un NOTIFY por el diálogo que se estaba intentando transferir, indicando en el cuerpo la respuesta de error con que ha contestado el objetivo de la transferencia, y que implica el final tanto de la transferencia como de la suscripción iniciadas por el REFER. La respuesta 200 OK del NOTIFY no se procesa.
- Desuscribirnos del diálogo que estaba siendo transferido, y volver a suscribir al SBB padre. No ha cambiado el diálogo de la pata del B2BUA sobre la que se intentó una transferencia, y la sesión continúa igual que antes de que se intentase. De esta forma el B2BUA SBB queda pendiente de la solicitud de nuevas transferencias, o de la liberación definitiva de la sesión.

5.2.2.3 El objetivo de la transferencia no contesta

Al enviar el INVITE dirigido al objetivo de la transferencia, el Push Transfer SBB hace uso de la *Timer Facility* del SLEE para fijar un temporizador, y cuyo valor es el máximo tiempo que se está dispuesto a esperar a que el objetivo de la transferencia la acepte. Este valor, que es configurable en el Mobility Manager, deberá ser inferior al que se establezca para la suscripción del REFER, de manera que no sea necesario refrescarla.

Si se produce alguno de los dos escenarios anteriores, el temporizador se cancelará al recibir la correspondiente respuesta del objetivo de la transferencia. En caso de que no se reciba contestación en el tiempo especificado, el temporizador finalizará, lanzándose en este caso un evento que será recogido en el Push Transfer SBB que esté tratando la transferencia. Lo que se hace al recibir este evento es crear un CANCEL para el INVITE, y enviarlo al objetivo de la transferencia, de manera similar a como se hace en 4.1.2 cuando se recibe un CANCEL del llamante y hay que cancelar el diálogo con el llamado.

Tras el envío de este CANCEL se recibirá una respuesta 200 OK correspondiente a dicho CANCEL, y que no se trata, y una repuesta de error 487 Request Terminated para el INVITE. Con el procesamiento de esta última respuesta de error termina su ejecución el Push Transfer SBB, y termina a su vez de forma no satisfactoria el intento de transferencia de sesión. Las acciones que se realizan para procesar esta respuesta son exactamente las mismas que en el escenario anterior, 5.2.2.2.

5.2.3 Interacción con el Transparency SBB

La interacción con el Transparency SBB va a ser muy similar a la vista en 5.1.3, con la diferencia de que aquí no habrá que realizar la asunción de que tiene que haber una oferta de códecs compatibles con los ya establecidos en la sesión. Ahora la oferta de códecs se hará desde el Mobility Manager, e irán únicamente los códecs ya negociados y que se encuentran establecidos en la sesión.

La primera ocasión en que se interactúa con el Transparency Manager es a la hora de construir y enviar el INVITE hacia el objetivo de la transferencia. Se invoca de forma síncrona al Transparency SBB indicándole en cuál de los diálogos se está produciendo la transferencia, y se espera que éste devuelva el SDP a incluir en el cuerpo del INVITE. La implementación de este

método síncrono en el Transparency SBB es prácticamente la misma que para el caso de la transferencia de tipo *Pull*, con la diferencia de que en este caso estamos construyendo la oferta SDP en lugar de la respuesta. Se construye un SDP a imagen del que se creó para la respuesta 200 OK que significó el establecimiento correcto de la sesión, y que contiene los códecs ya negociados y establecidos. Si la transferencia es en el diálogo saliente hay que cambiar las direcciones IP y puertos por las que usa el Transparency Manager en ese extremo. En la Figura 15 se mostró el SDP que iría en el INVITE del ejemplo con el que se ha ido ilustrando el funcionamiento.

Como ya se ha dicho, al enviar desde el Mobility Manager la oferta de códecs estos son los que se están usando en la sesión, si el objetivo de la transferencia no los soportase lo que hará será responder con una respuesta de error, dando como resultado el escenario 5.2.2.2. En caso de aceptar la oferta responderá con esos mismos códecs, los únicos que se ofrecen, indicando las direcciones IP y puertos que utilizará para comunicarse.

Es en el procesamiento de la respuesta 200 OK que conlleva la realización correcta de la transferencia cuando aparece la segunda y última interacción con el Transparency SBB. La invocación es exactamente la misma que se hace para las transferencias de tipo Pull, indicando en este caso el SDP recibido en la respuesta. Esta interacción permite que el Transparency Manager actualice su información y los datos se redirijan en la sesión de forma correcta entre los dos extremos.

Capítulo 6

Transparency Manager

El Transparency Manager es muy similar en concepto al Mobility Manager, pero aplicado al flujo de datos en lugar de al flujo de señalización. Su objetivo es situarse entre ambos extremos de una sesión correctamente establecida, y comportarse para cada uno de ellos como si fuese el extremo asociado con el que está manteniendo la sesión; de esta forma, todo lo que recibe de un extremo lo reenvía hacia el otro, y viceversa.

Si uno de los extremos desea moverse a una nueva localización, lo que hace el Transparency Manager es reenviar los datos recibidos del otro extremo de la sesión a esta nueva localización, y de manera simétrica los datos recibidos de la nueva localización mandarlos al otro extremo. El extremo que no se ha movido sigue enviando y recibiendo los datos desde el mismo sitio, el Transparency Manager, que es el extremo lógico contra el que tiene establecida la sesión. Esto es lo que se está denominando a lo largo del documento como transparencia de los datos.

En este capítulo del proyecto se explicará cómo se ha implementado esta funcionalidad, y se hará al igual que en los capítulos anteriores referentes al Mobility Manager, siguiendo los pasos de cómo se ha ido desarrollando el módulo. Para ello se explicará en primer lugar cómo se desarrolló la idea inicial que se tenía para el módulo, y en segundo y último lugar como un problema surgido con la idea inicial hizo que fuese necesario un cambio en la implementación.

6.1 Implementación inicial con click

Para la implementación se supone de antemano que los datos que intercambiarán los extremos de la comunicación en las sesiones que establezcan serán siempre flujos RTP [13], y que dichos flujos irán sobre el protocolo de transporte UDP.

El Transparency Manager lo que hará será recibir los paquetes de datos RTP de cada uno de los extremos, y reenviarlos al extremo asociado. Para conseguir este comportamiento se pensó en utilizar la herramienta *Click Modular Router* (click) [23]. Esta herramienta es básicamente un router software que permite trabajar con los paquetes de datos que se reciben o van a ser enviados por la red, y para ello proporciona una serie de módulos ya programados que permiten una gran variedad de funcionalidades, y permite también que dichos módulos sean modificados o que directamente se creen nuevos.

La idea inicial consistía en utilizar unos módulos ya programados que permiten sobrescribir las direcciones IP y puertos de los paquetes que se reciben. Cada uno de los extremos de la sesión envía paquetes de datos desde una IP y puerto origen determinados a una IP y puerto del Transparency Manager. Al recibir estos paquetes, el Transparency Manager sobrescribe la

dirección IP y puerto origen por las suyas propias, y los reenvía a la dirección IP y puerto del extremo asociado. La Figura 16 muestra un esquema explicativo.

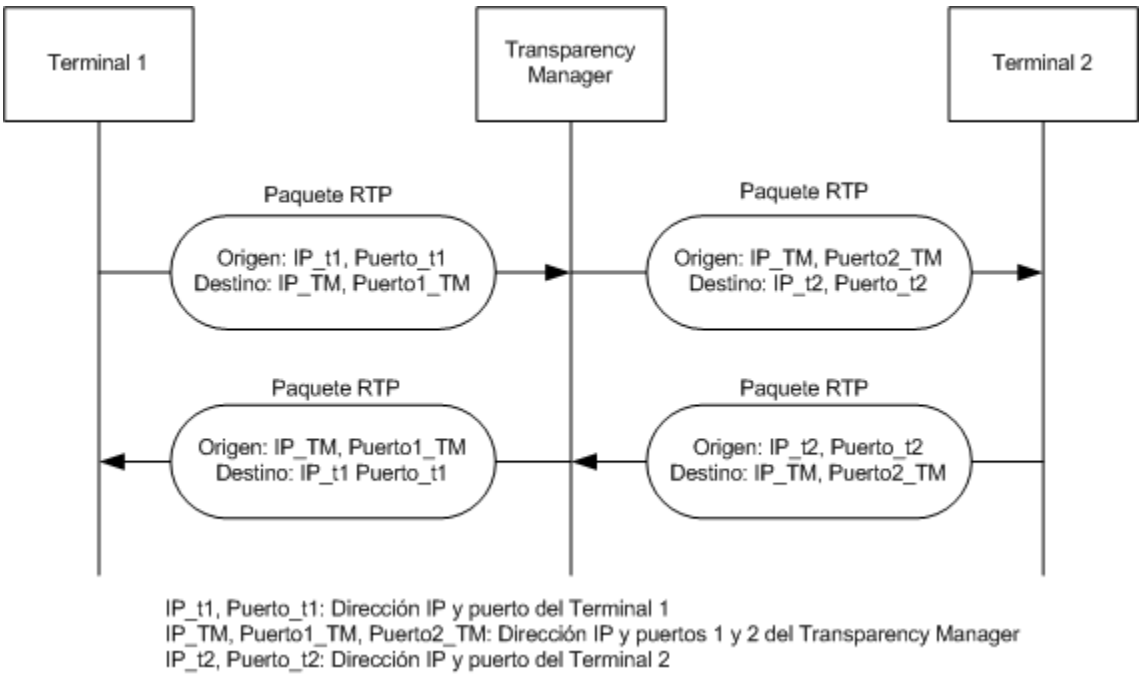

Figura 16: Funcionamiento del Transparency Manager

Para conseguir este comportamiento con click es necesario mantener actualizado un fichero de configuración que refleje el comportamiento que se quiere, básicamente los cambios de direcciones IP y puertos que hay que hacer sobre los paquetes, a los que se les denominará como mapeos. Cada vez que haya una nueva sesión de datos, o que se produzca una transferencia de sesión, hay que actualizar el fichero para añadir o modificar la configuración, y al mismo tiempo es necesario hacer una llamada al sistema operativo para que click recoja los cambios del fichero.

Este comportamiento es el que lleva al diseño pensado e implementado para el Transparency Manager, y que se describirá en detalle a continuación.

6.1.1 Patrón de diseño

Dados los requisitos de tener actualizado siempre un fichero de configuración, y de hacer llamadas al sistema operativo, se elige un patrón de diseño *Singleton* que además contenga métodos sincronizados. De esta forma se tendrá una única instancia de clase que mantendrá en todo momento el estado del Transparency Manager, que consistirá en los mapeos correspondientes a las sesiones activas, y que ejecutará de forma síncrona las llamadas que se le hagan para cambiar su estado, evitando de esta forma que se produzcan situaciones de carrera.

La única instancia de clase será el objeto que se ofrezca remotamente por RMI al Mobilty Manager, tal y como se ha visto en 4.2. Si se cambiase la forma de invocar habría que cambiar la forma en que se ofrece la interfaz de comunicación, pero se mantendría la idea de tener una única instancia de clase con todo el estado que se invocase de forma síncrona.

6.1.2 Diagrama de clases

Para mantener el estado en el Transparency Manager se elabora un diagrama de clases, manteniendo la filosofía orientada a objetos de Java. Serían las siguientes:

- ***`Endpoint`***. La clase básica que identificará los puntos de comunicación, tanto de los extremos como del propio Transparency Manager. Estará formada por la IP y el puerto que se usarán para el intercambio de los datos RTP.
- ***`SessionInfo`***. La clase ya vista 4.2.1, un mapa con los distintos `Endpoint` que usarán los extremos y el Transparency Manager para comunicarse. Para el servicio de movilidad ofrecido en el proyecto será la forma de recoger el contenido que irá en el SDP. El hecho de no ser directamente el SDP permite que el Transparency Manager sea algo más genérico, y que se pueda utilizar para otros proyectos que requieran transparencia de datos de una forma similar.
- ***`Flow`***. Representará un flujo de datos, y que irá entre dos `Endpoint`: uno perteneciente a un extremo de la comunicación y el otro al Transparency Manager.
- ***`EarlySession`***. Un mapa que recogerá los `Flow` a establecer entre uno de los extremos de la comunicación y el Transpareny Manager. El número de `Flow` dependerá de los flujos RTP que se pretenda establecer, uno si es sólo audio, dos si es audio y vídeo,…
- ***`Mapping`***. Estará formado por los dos `Flow` que representarán la comunicación extremo a extremo, y que estarán relacionados. Lo que se reciba en el Transparency Manager desde uno de los extremos (representa uno de los `Flow`), se enviará desde el propio Transparency Manager al otro de los extremos (representa el `Flow` asociado), y viceversa.
- ***`Session`***. Un mapa que en este caso recogerá los distintos `Mapping` que conformarían una sesión correctamente establecida entre dos extremos.

La instancia de clase que se ofrece por RMI tendrá tres atributos de clase que recogerán el estado del Transparency Manager en todo momento:

- Una lista con los puertos que tiene disponibles para utilizar, y cuyo número limitará en última instancia el número de sesiones que se pueden tener establecidas. Se denominará como ***`ports`***.
- Un mapa de objetos `EarlySession`, que recogerá las sesiones que se pretenden establecer. Estarán indexadas por un id único. Se denominará como ***`earlySessions`***.
- Un mapa de objetos `Session`, que recogerá las sesiones que se tienen establecidas de forma correcta, y que estará indexado por el mismo id único que el mapa anterior. Se denominará como ***`sessions`***.

6.1.3 Métodos ofrecidos en el interfaz RMI

Aquí se analizarán más en detalle el funcionamiento de los métodos ofrecidos remotamente por RMI, y que ya fueron presentados en 4.2.1.

- `SesionInfo addEarlySession(String id, SessionInfo userAgenteSessionInfo)`: El objetivo es hacer en el Transparency Manager una reserva de recursos previa a un establecimiento de sesión. Uno de los extremos, al que se denominará en adelante como llamante, desea establecer una sesión. A la hora de invocar se indican las direcciones IP y puertos que usaría el llamante para los flujos RTP que desea establecer, y se espera recibir las que se usarían en el Transparency Manager. Los pasos a realizar son:
    - Se comprueba si se es posible hacer la reserva. Para ello se verifica que el identificador sea único y que no pertenezca ya a una reserva previa o a una sesión confirmada, y también que haya suficiente número de puertos disponibles para que la sesión pueda establecerse.
    - Se prepara un objeto `EarlySession`, que almacenará el estado de la reserva en el Transparency Manager, y un objeto `SessionInfo` con la información a devolver. Se recorren los distintos `Endpoint` recibidos y para cada uno de ellos:
        - Se construye un `Endpoint` para el Transparency Manager, que sería por el que se reenviarían los datos hacia el destino, el punto de comunicación con el posible otro extremo de la sesión. La dirección IP del `Endpoint` será la de la máquina donde estará instalado el Transparency Manager, y el puerto uno disponible de la lista.
        - Se añade el `Endpoint` al objeto `SessionInfo` a devolver.
        - Se crea un objeto `Flow` a partir de cada `Endpoint` recibido y del construido, y se añade al objeto `EarlySession`.
    - Se añade el objeto `EarlySession` al mapa `earlySessions`, y se devuelve el objeto `SessionInfo`.

- `removeEarlySession(String id)`: Al invocar este método se indica que la reserva previa hecha con id único determinado no se va a completar, y no va a establecerse una sesión. Los pasos a realizar en este método son:

    - Obtener de `earlySessions` el objeto `EarlySession` con la reserva previa.
    - Se recorre cada `Flow` que compone el objeto `EarlySession`, y se devuelve a la lista de puertos el puerto que se reservó para el `Endpoint` del Transparency Manager, de forma que pueda volver a usarse.

- `SessionInfo addConfirmedSession(String id, SessionInfo userAgentSessionInfo)`: Se va a establecer finalmente una sesión para la que previamente se habían reservado recursos. Se indican en la invocación las direcciones IP y puertos a usar por el otro extremo de la sesión, al que se denominará en adelante como llamado, y se espera recibir las que se usarán en el Transparency Manager. A partir de esta invocación

habría que poner en funcionamiento el reenvío de los paquetes, la denominada transparencia de los datos. Los pasos que se realizan son:

- Se obtiene el objeto `EarlySession` que contiene la reserva previa. Se prepara un objeto `Session` que contendrá todo la información de la sesión, y el objeto `SessionInfo` con la información a devolver.
- Se recorren los distintos `Endpoint` recibidos, correspondiente cada uno a uno de los tipos de flujo RTP que se quieren establecer. Para cada uno de ellos:
  - Se construye un `Endpoint` para el Transparency Manager al igual que se hizo en el método `addEarlySession`, y que en este caso representará el punto de comunicación con el llamante. Se añade al objeto a devolver.
  - Se obtiene el objeto `Flow` de la `EarlySession`, al que se denominará flujo entrante. El `Endpoint` del Transparency Manager que contiene se utiliza para crear un nuevo objeto `Flow` junto al `Endpoint` recibido, que se denominará flujo saliente. Al flujo entrante se le cambiará el `Endpoint` del Transparency Manager por el que se acaba de construir en el punto anterior.
  - Ya se tiene toda la información que conformaría una sesión, recogida en el flujo entrante (que conforman el llamante y el Transparency Manager), y el saliente (que conforman el llamado y el Transparency Manager). Se crea un mapeo con estos dos flujos y se añade al objeto `Session`.
- Se añade el objeto `Session` al mapa `sessions`. En este momento se tiene un cambio en el estado de las sesiones, se recorrería todo el mapa para construir el fichero de configuración de click, y posteriormente se haría la llamada al sistema que permite que se produzca la transparencia de los datos para las sesiones activas.
- Se devuelve el objeto `SessionInfo`.

- `removeConfirmedSession(String id)`: Se indica al Transparency Manager que una sesión establecida ha finalizado. Los pasos a realizar son:
  - Se obtiene del mapa `sessions` el objeto `Session` con la información de la sesión.
  - Se recorre cada uno de los mapeos que conforman la sesión. De cada mapeo se obtienen los dos flujos que lo componen, y de cada flujo el `Endpoint` del Transparency Manager que lo conforma. Se devuelve a la lista de puertos el puerto del `Endpoint`, para que pueda usarse para posteriores sesiones.

- `transferSession(String id, SessionInfo userAgentSessionInfo, boolean incomingTransfer)`: Uno de los dos extremos de la sesión se ha movido, y como resultado ha cambiado los `Endpoint` que utilizará para los flujos RTP. Se invoca al Transparency Manager con la nueva información, y se indica a su vez si el extremo que se ha movido es el llamante o el llamado. Los pasos a realizar son:

- Se obtiene del mapa `sessions` el objeto `Session` correspondiente.
- Se recorren los distintos `Mapping` que conforman la sesión, y de cada uno de ellos se obtiene el flujo entrante o el saliente, según sea el extremo que se ha movido. En dicho flujo, se cambia el `Endpoint` que representa al llamante o el llamado con la correspondiente información recibida.
- Al haberse actualizado la información de las sesiones, se vuelve a interaccionar con click de la misma manera a como se hizo en `addConfirmedSession`, con el objetivo de conseguir la transparencia de los datos.

El Transparency Manager no realiza una comprobación exhaustiva de posibles errores, y confía en que se le invoque de forma correcta. Esta forma correcta consiste en que si se confirma una sesión, se transfiere o se elimina, la sesión realmente exista; o que el número de `Endpoint` que se indican al confirmar o transferir una sesión corresponda exactamente con el número que previamente se indicó que dicha sesión comprendería. En el proyecto la responsabilidad de invocar de forma correcta recae en el Transparency SBB del Mobility Manager, 4.2.

6.2 Cambio en la implementación, uso de sockets

Una vez desarrollado y probado de forma independiente el funcionamiento del Transparency Manager, realizando invocaciones controladas desde un programa Java por RMI, se pasa a probar la solución de manera integrada. Para ello se establecen sesiones entre usuarios registrados en el Core IMS, invocándose el Transparency Manager directamente desde el Mobility Manager al establecerse dichas sesiones, y al producirse transferencias en ellas.

Se observa que, de manera aleatoria y sin seguir ningún patrón, hay veces en que la llamada a sistema para instalar el fichero de configuración de click no se completa, y la máquina que aloja el Transparency Manager deja de responder, necesitando de un reinicio para volver a estar operativa. Esto hace que todo el sistema se resienta, pues aunque el intercambio de señalización SIP siga funcionando, no hay intercambio de datos entre los extremos.

Tras un período de investigación se concluye que es un fallo en los módulos de click que se están utilizando, y que es un problema conocido del que aún no se tiene solución, al ser click una herramienta experimental y en continuo desarrollo. Para eliminar el problema de tener en la solución elegida una fuente de error no controlada, se decide optar por una alternativa.

La alternativa escogida consiste en utilizar los sockets de Java para comunicaciones UDP. Se usarán sockets para recibir los paquetes RTP de cada uno de los extremos, y para reenviarlos al extremo correspondiente. Dado el trabajo invertido en la construcción de la solución ya explicada, se decide adaptar toda la estructura a la utilización de sockets.

6.2.1 Cambios en la estructura

El primer cambio consistirá en eliminar toda la interacción con click, que como se ha visto en 6.1.3 se daba en los métodos `addConfirmedSession` y `transferSession`, y que consistía en construir un fichero de configuración con toda la información de las sesiones activas y realizar una llamada a sistema operativo.

Se va a cambiar el objeto `Mapping`. Además de tener la información de los flujos entrante y saliente ya explicados, contendrá dos hilos. Estos hilos serán los encargados de recibir y reenviar los paquetes para proporcionar la transferencia, y cada uno se encargará de un sentido de la comunicación. De esta forma cada hilo recibirá los paquetes de uno de los extremos de la comunicación, el llamante o el llamado, y los reenviará al otro extremo, el llamado o llamante respectivamente.

Se implementarán tres métodos en la clase `Mapping`, que se llamarán desde los métodos ofrecidos en la interfaz RMI. Se describirá a continuación la función de cada uno de los métodos, que lo que harán será interactuar con los dos hilos que forman parte del objeto, y cuyo funcionamiento específico se detallará en el siguiente subapartado. Los métodos son los siguientes:

- `startForwarding(int bufferSize)`: Desde el método `addConfirmedSession`, cuando se construye el objeto `Mapping`, se invoca a este método. El entero que se indica es el tamaño máximo de los paquetes que se esperan recibir, que será un valor configurable y al que se dará el valor de 8.192bytes [26].
  Lo que se hace en este método es crear los dos hilos con la información de los flujos que se tiene, e iniciarlos.
- `stopForwarding()`: Se invoca a este método desde `removeConfirmedSession`, cuando se van recorriendo los distintos mapeos que componen una sesión que se ha terminado.
  Lo que se hace es invocar en cada uno de los hilos un método que termine el reenvío de los paquetes y haga que el hilo finalice.
- `updateForwarding(boolean incomingTransfer)`: Se invoca a este método desde `transferSession`, una vez que se actualiza el flujo en que se ha producido la transferencia al ir recorriendo los distintos `Mapping`. Se indica el extremo en que se ha producido la transferencia.
  El comportamiento depende del extremo en que se produzca la transferencia, y lo que se hace es:
  - En el hilo que recibe los paquetes desde el extremo que no se ha movido y los reenvía al que sí lo ha hecho, actualizar el destino al que se reenvían dichos paquetes.
  - En el otro de los hilos, se actualiza el extremo desde el que se espera recibir los paquetes.

6.2.2 Hilo para el reenvío de paquetes

Como ya se ha indicado, se tendrá un hilo para cada uno de los sentidos de la comunicación. Este subapartado detallará cómo es su funcionamiento, y para ello analizarán las distintas fases de su ciclo de vida.

6.2.2.1 Creación del hilo

El hilo se crea invocando a su constructor, en él se le pasarán dos objetos `Flow` que indicarán de dónde tiene que recibir los paquetes de datos (flujo entrante), y a dónde tiene que reenviarlos (flujo saliente). También se le indica el tamaño máximo de paquete que se podría recibir. Las labores que se llevan a cabo en el constructor son las siguientes:

- Se crea un socket al que se denominará socket entrante. Este socket se abre en el `Endpoint` (dirección IP y puerto) del Transparency Manager correspondiente al flujo entrante. A su vez, el socket se conecta al `Endpoint` correspondiente al extremo de la comunicación del mismo flujo entrante, pues sólo se quieren recibir paquetes que se envíen desde este extremo.
- Se crea un nuevo socket, al que se denominará saliente. Se abre en el `Endpoint` del Transparency Manager correspondiente al flujo saliente. En este caso se guarda la dirección a la que enviar en primera instancia los paquetes, que será el `Endpoint` correspondiente al extremo de la comunicación, y que podrá cambiar si el extremo se mueve.

6.2.2.2 Arranque y funcionamiento del hilo

Cuando se llama al método de arranque del hilo, este entra en un bucle de funcionamiento controlado por un booleano. Cuando el valor del booleano cambia y se sale del bucle, termina su ejecución.

Dentro del bucle, la ejecución del hilo queda suspendida esperando recibir un paquete de datos por el socket entrante. Una vez que se recibe un paquete de datos:

- Se crea un nuevo paquete de datos, copiando el contenido del recibido.
- Se envía el nuevo paquete desde el socket saliente, poniendo como destino la dirección que se tiene almacenada.

Al enviar el paquete desde el socket saliente, se actualiza el origen del paquete con la información del `Endpoint` que se está utilizando en el Transparency Manager.

La orden de recibir un paquete de datos queda el hilo suspendido. Para salir de este estado es necesario que se lance una excepción. Al capturar la excepción hay que hacer un procesamiento para comprobar qué es lo que ha pasado.

Si la excepción es que el destino al que se ha querido enviar el paquete no es alcanzable, es muy probable que el extremo de la comunicación aún no esté preparado para recibir los paquetes. Aún así hay que seguir intentando el reenvío de todos los paquetes que se reciban, pues se ha visto al probar el servicio que muchas veces uno de los extremos empieza a mandar datos antes de que el otro esté preparado para recibirlos.

Si la excepción es que el socket se ha cerrado, se debe a alguna de las causas que se analizarán en los dos siguientes subapartados, en ellos se verá que es lo que se hace en cada caso.

6.2.2.3 Parada del hilo

Cuando ya no se van a reenviar más paquetes, lo que hay que hacer es cerrar el socket entrante para que no se quede bloqueado esperando recibir un nuevo paquete de datos, con lo que se lanza una excepción.

Al capturar la excepción, lo que se hace es cambiar el booleano del bucle visto en 6.2.2.1 para que salga de él, y el hilo pueda terminar su ejecución y ser eliminado de forma ordenada.

6.2.2.4 Manejo de la transferencia

Como se ha visto en 6.2.1, según qué sentido de la comunicación esté tratando el hilo el comportamiento será distinto.

Si cambia el extremo al que se reenvían los paquetes, lo que se hace es invocar un método que cambia la dirección de este extremo. De esta forma, en el siguiente paquete de datos que se reciba la dirección destino que se pondrá al paquete a reenviar será la nueva, y seguirá siendo ésta para los sucesivos paquetes.

Si lo que cambia es el extremo desde el que se reciben los paquetes, el procedimiento es un poco más complejo. Se invoca un método del hilo en que se recibe el nuevo objeto `Flow`, en el que ha cambiado el `Endpoint` del extremo de la comunicación. Los pasos que se siguen son los siguientes:

- Se activa un booleano, para indicar que ha habido una transferencia, y se guarda el objeto `Flow` recibido.
- Se cierra el socket entrante para que se lance una excepción mientras el hilo está esperando recibir un nuevo paquete de datos.
- Al procesar la excepción se consulta el booleano. Si no está activo lo que se tiene es el comportamiento visto en 6.2.2.3.
- Al haberse producido una transferencia el booleano está activo. Lo que se hace es crear nuevamente el socket entrante tal y como se hizo en 6.2.2.1, y se desactiva el booleano indicando que ha habido una transferencia. La ejecución del hilo vuelve a detenerse a la espera de recibir un paquete, pero ahora procedente de la nueva dirección, que es a dónde se ha conectado el socket al crearse nuevamente.

Capítulo 7

Pruebas

Este capítulo recogerá los procedimientos llevados a cabo en el proyecto para probar y validar la solución propuesta e implementada.

Tal y como se ha ido describiendo en los capítulos anteriores, el proceso de desarrollo ha sido incremental, añadiéndose poco a poco las distintas funcionalidades. Esto ha hecho que el proceso de prueba haya sido también incremental, probándose cada nueva funcionalidad desarrollada hasta validar su correcto funcionamiento, y antes de continuar con el siguiente paso en el desarrollo.

La herramienta utilizada para las pruebas ya ha sido presentada en 3.4, SIPp. El procedimiento para probar el funcionamiento ha sido siempre el mismo en todas las fases del desarrollo:

- Creación de escenarios SIPp con el tráfico de señalización (y datos si también se están probando) que se desea obtener.
- Ejecución controlada de los escenarios, usando el depurador Java del Entorno Integrado de Desarrollo Eclipse [24], que ha sido el empleado durante el desarrollo.
- Comprobación en los archivos de log de que el comportamiento ha sido el correcto.
- Comprobación de los mensajes de señalización (y datos si es el caso) usando el analizador de tráfico de red Wireshark [25].

En los siguientes subapartados del capítulo se irán describiendo las diferentes pruebas realizadas, acorde con el ciclo de desarrollo del proyecto, y tomando siempre como parte central del proyecto el Mobility Manager.

7.1 Funcionamiento en modo B2BUA

La primera fase del desarrollo consistió en tener desarrollado en el Mobility Manager una aplicación que funcionase como un B2BUA de SIP, y fuese capaz de controlar el establecimiento y liberación de sesiones tal y como se ha visto en 4.1.

Se crean escenarios SIPp que permitan probar todos los casos ya explicados:

- ***Establecimiento correcto de sesión y liberación desde ambos extremos***. Se crean dos escenarios para el usuario que inicia la llamada, el denominado como llamante, y dos para el que la recibe, el llamado. En la primera pareja de escenarios el llamante es el que libera la sesión mandando un BYE, mientras que en la segunda es el llamado el que lo hace. El último caso es importante, ya que se cambia el contenido de las cabeceras From y To.

- ***Fallo en el establecimiento de la sesión, el llamado rechaza***. Se crea un escenario para el llamado en el que contesta al INVITE con una respuesta 486 Busy Here. Un nuevo escenario para el llamante espera recibir dicha repuesta. El funcionamiento sería el mismo en caso de que el llamante contestase con una repuesta de error distinta.
- ***Fallo en el establecimiento de sesión, el llamante desiste***. Se crea un nuevo escenario para el llamante que envíe un CANCEL tras un período de tiempo, y espere recibir una repuesta 200 OK para el CANCEL y una respuesta 487 Request Terminated para el INVITE previo. A su vez se crea un escenario complementario para el llamado, que espera recibir el CANCEL, y contesta con las dos respuestas descritas anteriormente.

Para probar estos escenarios establecemos una comunicación directa entre los scripts de SIPp y el Mobility Manager. El llamante manda los mensajes directamente al Mobility Manager, y el Mobility Manager directamente al llamado. El Mobility Manager aparece como llamado para el llamante, conformando la primera de las patas del B2BUA, y como llamante para el llamado, conformando la segunda de las patas.

Se comprueba que el Mobility Manager cree de forma correcta el nuevo diálogo con el llamado, y reenvíe de forma correcta los mensajes que recibe de cada una de las patas por la complementaria.

Utilizando las opciones que permite SIPp, se prueba a realizar muchas llamadas simultáneas, así como a ejecutar en paralelo diversos escenarios para el llamante y el llamado. Todas las pruebas se realizan de forma satisfactoria, y el Mobility Manager es capaz de procesar sin errores del orden de decenas de llamadas por segundo, utilizándose para las pruebas un ordenador básico con procesador de doble núcleo y 4 GB de RAM.

Los escenarios utilizados para esta serie de pruebas son los que aparecen en B.4.1.1.

7.2 Integración con el Transparency Manager

Una vez verificado el comportamiento en modo B2BUA del Mobility Manager, lo siguiente consiste en probar la integración con el Transparency Manager, todo el desarrollo descrito en el apartado 4.2.

Mientras que en el subapartado anterior se comprobaba que los mensajes SIP se creasen y modificasen de forma correcta, en este subapartado lo que hay que comprobar es que la carga SDP tanto del INVITE dirigido al llamado como de la respuesta 200 OK dirigida al llamante se modifican correctamente para que el flujo de datos entre los dos extremos de la comunicación pase a través del Transparency Manager.

Los escenarios SIPp a utilizar serían los mismos que en el subapartado anterior, únicamente modificándolos para usar la capacidad de poder enviar flujos RTP o de hacer eco de los que se reciban, pudiéndose así comprobar si el flujo de datos se redirige de forma correcta a través del Transparency Manager.

Para las pruebas se envía un flujo RTP desde el llamante al llamado, que a su vez hace eco del flujo recibido. Se comprueba observando en Wireshark si los paquetes de datos se redirigen correctamente a través del Transparency Manager y acorde a la información que aparecen en la carga SDP de los mensajes SIP. Se prueba al igual que en el subapartado anterior a realizar llamadas simultáneas, y se siguen pudiendo procesar del orden de decenas de llamadas por segundo, procesándose de forma correcta tanto el flujo de señalización como el flujo de datos.

En B.4.1.2 aparecen los escenarios utilizados.

7.3 Integración en el Core IMS

Una vez se tiene la aplicación funcionando en modo B2BUA y con el tráfico de datos redirigido a través del Transparency Manager, el siguiente paso consiste en integrar el Mobility Manager dentro del Core IMS. Se probarán los mismos casos que en los subapartados anteriores, pero verificando ahora que los mensajes de señalización se enrutan correctamente a través del Core.

La configuración necesaria es la que aparece en 3.1, y que se ve en imágenes en B.1. La ventaja de tener ya integrado el Mobility Manager en el Core es que se puede probar la aplicación usando directamente los *softphones* myMonster e IMSDroid presentados en 3.4. Se verifican los tres posibles escenarios de movilidad que se pueden dar, y que se enumeran a continuación:

- ***Movilidad en origen:*** El usuario llamante tiene el *Service Profile* de movilidad, por lo que todos los INVITE que origine se enrutan hacia el Mobility Manager. El usuario llamado es un usuario sin movilidad.
  Cuando el llamante inicia la llamada, envía el INVITE a su P-CSCF (se le puede denominar origP), que lo dirige a su S-CSCF (se le puede denominar origS). Éste último comprueba los *Initial Filter Criteria*, y al tener activo el servicio de movilidad dirige el INVITE al Mobility Manager. El Mobility Manager devuelve el INVITE al origS, pero habiendo creado un nuevo diálogo tal y como se vio en 4.1, el origS comprueba que no se cumple ningún otro *Initial Filter Criteria* y comienza a enrutar hacia el destino de la llamada.
  Al estar el llamante registrado en el mismo Core IMS, origS redirige el nuevo INVITE hacia el S-CSCF del llamado (se le puede denominar termS), termS lo entrega al P-CSCF del llamado (se le puede denominar termP), y termP en última instancia lo dirige ya al llamado.
  Así quedarían establecidas las dos patas del B2BUA, la entrante formada por llamante-origP-origS-Mobility Manager, y la saliente por Mobility Manager-origS-termS-termP-llamado. Aquí radica la diferencia fundamental con las pruebas realizadas en 7.1, los mensajes INVITE siguen una serie de rutas a través del Core reflejadas en las cabeceras Route y Record-Route de los mensajes, mientras que antes la conexión era directa entre los extremos y el Mobility Manager. En la Figura 17 se puede observar un esquema.

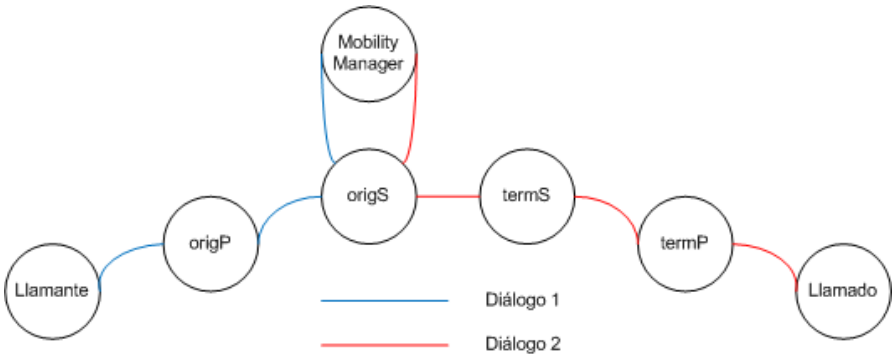

Figura 17: Movilidad en origen

Se prueba a establecer correctamente la sesión liberándola desde ambos extremos, a que el llamado rechace, y a que el llamante desista, verificando que la señalización sea correcta, y que el flujo de datos se redirija a través del Transparency Manager.

- ***Movilidad en destino:*** Ahora es el llamado el que tiene activo el servicio de movilidad, por lo que los INVITE que vayan dirigidos hacia él se enrutarán hacia el Mobility Manager.
  El INVITE del llamante irá desde origP a origS, y de aquí a termS. Se evaluarán los *Initial Fliter Criteria* y se enrutará el INVITE hacia el Mobility Manager. Éste último lo devolverá a termS como un INVITE perteneciente a un nuevo diálogo, y llegará al llamado pasando previamente por el termP.
  La pata entrante del B2BUA ahora la forman llamante-origP-origS-termS-Mobility Manager, mientras que la saliente la forman Mobility Manager-termS-termP-llamado.
  Se comprueban los mismos supuestos que en el escenario anterior, verificando que en este caso los mensajes SIP siguen las nuevas rutas. Así mismo se verifica también que el flujo de datos se redirija a través del Transparency Manager. La Figura 18 muestra un esquema.

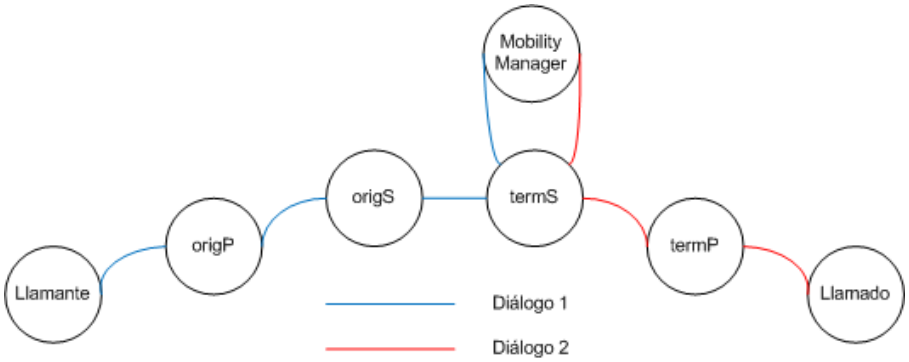

Figura 18: Movilidad en destino

- ***Doble movilidad:*** En este último caso tanto el usuario llamante como el llamado tienen activo el servicio de movilidad. Este es el caso más complejo desde el punto de señalización SIP, pues implicará dos veces al Mobility Manager, pero se puede ver como la simple combinación de los dos anteriores.

El llamado enviará el INVITE, que atravesará origP, origS y llegará al Mobility Manager al comprobarse el *Initial Filter Criteria*. El Mobility Manager devolverá a origS un INVITE perteneciente a un nuevo diálogo, y origS lo entregará a termS. Hasta aquí tendríamos el mismo caso que en movilidad en origen.
Ahora termS evaluará los *Initial Filter Criteria*, enrutando el INVITE nuevamente al Mobility Manager. El Mobility Manager interpreta este INVITE como un nuevo INVITE que recibe, sin tener ninguna relación con el anterior, y dando lugar a una nueva y totalmente independiente ejecución del servicio. Crea un nuevo INVITE que entrega a termS, y que en última instancia llega al llamado pasando por termP. Es el mismo caso que en movilidad en destino.
Tenemos por tanto dos B2BUA interactuando en la sesión. En el primero de ellos la pata entrante la forman llamante-origP-origS-Mobility Manager 1, y la saliente Mobility Manager 1-origS-termS-MobilityManager2. Para el segundo la pata entrante es la misma que la saliente del primero, y su saliente la forman MobilityManager2-termS-termP-llamado. Se muestra un esquema en la Figura 19.

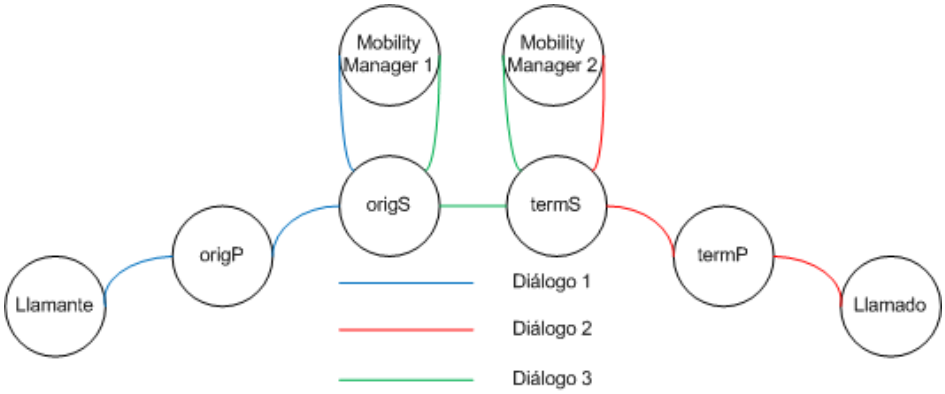

Figura 19: Doble movilidad

Se consigue de esta forma poder dar el servicio de movilidad a ambos usuarios, y que ambos puedan realizar transferencias de sesión sin que el otro extremo sea consciente de ello. Se comprueban los mismos supuestos de establecimiento, liberación y cancelación de sesión que en los casos anteriores, verificando que la señalización es correcta.
A su vez se comprueba el flujo de datos, que en este caso atraviesa dos veces el Transparency Manager. Los paquetes que envíe el llamante van al Transparency Manager, que se los vuelve a enviar a sí mismo antes de que lleguen al llamado, ocurriendo lo equivalente en el caso inverso. Mientras que en el caso de la señalización SIP si es lógico y correcto que pase dos veces por el Mobility Manager, para ser acorde a la idea de que los servicios en IMS se prestan individualmente para cada usuario, en el caso de los datos se podría comprobar cuándo se da la circunstancia de lo doble movilidad y evitar que el Transparency Manager se redirija los datos a sí mismo.
En el desarrollo de este proyecto se ha querido dejar la implementación de forma genérica, sin incluir esta lógica adicional, que es algo que se podría evaluar como posible futura mejora.

Todos los escenarios anteriores se prueban con myMonster e IMSDroid, pero así únicamente puede probarse una llamada a la vez. Se crean también escenarios SIPp que permitan probar llamadas simultáneas y ejecutar en paralelo diversos escenarios, tal y como se hizo en los subapartados anteriores.

Se crean nuevos escenarios para poder registrar usuarios en el Core IMS, y se modifican los creados anteriormente para adaptarse al hecho de estar integrados en dicho Core. Los nuevos escenarios aparecen en B.4.1.3.

Se prueba a hacer llamadas simultáneas, pero en el momento en que se pasa de una llamada por segundo el Core IMS empieza a fallar y a no poder procesar todos los mensajes. Esto puede deberse a estar utilizando una máquina virtual, y también a que dicha máquina virtual contiene una versión no muy actualizada de los componentes del Core y que puede no estar muy optimizada.

Aun así se cumple el objetivo de probar la solución implementada en un entorno totalmente integrado, y se llega a la conclusión de que se tiene un límite en la tasa de procesamiento de llamadas que lo da el Core IMS, y no la aplicación desarrollada para el Mobility Manager.

7.4 Realización de transferencias tipo *Pull*

Tal y como se vio en 5.1, para que puedan realizarse este tipo de transferencias debe de tenerse establecida de forma correcta una sesión entre dos usuarios. A partir de ahí si alguno de los dos usuarios tiene activo el servicio de movilidad, puede mandar un nuevo INVITE con una cabecera Replaces desde un nuevo terminal, con el objetivo de traerse a dicho nuevo terminal la sesión que tiene activa.

Las transferencias de tipo *Pull* pueden producirse para cualquiera de los tres tipos de movilidad vistos anteriormente, el resultado final consiste en que una de las dos patas del B2BUA, aquella que se mantiene con el usuario que tiene movilidad, se sustituye para establecer comunicación con el nuevo terminal. En el caso de la doble movilidad, en el que aparecen dos B2BUA distintos, sólo cambia una pata de aquél que sirve al usuario que se mueve, manteniéndose sin cambios el otro.

El nuevo INVITE con la cabecera Replaces llega al Mobility Manager dado que se produce desde un terminal del usuario con movilidad. El usuario envía el INVITE a su P-CSCF, de aquí va al S-CSCF, éste último evalúa el *Initial Filter Criteria* y lo enruta hacia el Mobility Manager, que realiza todo la lógica de la transferencia.

Este tipo de transferencias únicamente se pueden probar con escenarios SIPp, ya que los *softphones* myMonster e IMSDroid no envían mensajes INVITE con cabecera Replaces. Es también necesario conocer los identificadores del diálogo SIP que se quiere transferir, call-id y tags. Como se indicó brevemente en 5.1 se supone que debería haber algún mecanismo externo por el que el usuario pudiera conocer las sesiones que tiene activas y poder así realizar

este tipo de transferencias, pero en el proyecto no se ha contemplado la implementación de este mecanismo.

Lo que se hace para poder probar este tipo de transferencias es modificar el código del Mobility Manager de tal forma que guarde en un lugar conocido unos ficheros con la información de los diálogos. El Mobility Manager no sabe si está dando movilidad al usuario origen o al destino, por lo que está preparado para realizar transferencias en cualquiera de sus dos extremos, y guarda un fichero con la información de cada uno de ellos. Al ejecutar los scripts de SIPp se leen estos ficheros y se crea la cabecera Replaces con los valores correctos.

Se necesitan tres escenarios SIPp para poder probar este tipo de transferencias. Uno representa al llamante, que inicia la sesión mandando un INVITE, y otro al llamado, que recibe el INVITE; el tercero de los escenarios es el nuevo terminal, que puede ser del llamado o el llamante, y que manda el INVITE con la cabecera Replaces. En ese momento, uno de los dos escenarios anteriores termina al recibir un BYE del Mobility Manager, se ha producido la transferencia.

Se realizan diversas pruebas, transfiriendo el llamante y el llamado, una vez que se ha producido la transferencia colgando desde los dos extremos, realizando más de una transferencia, etc. Se verifica la señalización en todas ellas, y también se verifica que se produzca la transferencia de los datos: el extremo que no se mueve recibe los paquetes del nuevo terminal, pero le llegan desde el mismo punto de comunicación anterior, el Transparency Manager; a su vez este extremo que no se mueve continúa mandando los paquetes al Transparency Manager, pero ahora llegan al nuevo terminal.

Los escenarios usados aparecen en B.4.1.4.

7.5 Realización de transferencias tipo *Push*

Este tipo de transferencias es más sencillo de probar, ya que no se necesita ningún tipo de información de contexto adicional, y las pruebas se pueden realizar con el softphone myMonster, que soporta el envío de mensajes REFER.

Para este tipo de transferencias, y tal y como se ha visto en 5.2, un usuario con movilidad que tiene establecida una sesión decide llevársela a otro terminal donde esté registrado, y lo hace mandando un mensaje REFER desde el primero de sus terminales. Este mensaje llega directamente al Mobility Manager siguiendo las rutas ya establecidas para el diálogo SIP, y éste lo que hace es terminar el diálogo con el primero de los terminales y establecer uno nuevo con el segundo, sustituyendo así una de las dos patas del B2BUA.

Para terminar el primero de los diálogos, el Mobility Manager manda un BYE al antiguo terminal siguiendo las rutas establecidas para el diálogo SIP. El nuevo INVITE que origina para el nuevo terminal lo manda directamente al S-CSCF que le sirve usando como origen el PSI, tal y como se vio en 5.2.2. Al no tener el PSI ningún tratamiento especial en origen, se dirige el INVITE al S-CSCF que sirve al usuario destino; este usuario sí tiene un tratamiento especial, al

tener el servicio de movilidad, pero como el INVITE procede del PSI no cumple el *Initial Filter Criteria* y va directamente al P-CSCF, y de aquí al usuario.

Se prueba la aplicación realizando transferencias para el usuario llamante en el caso de movilidad en origen, para el llamado en el caso de movilidad en destino, y para ambos en el caso de la doble movilidad. Se prueba así mismo a realizar más de una transferencia, y a liberar posteriormente la sesión desde cualquiera de los extremos. Finalmente se prueban también los dos posibles casos de error en la transferencia que se describieron en 5.2.2: que el nuevo terminal rechace la transferencia, o que el nuevo terminal no conteste en un determinado intervalo de tiempo.

Se verifica que la señalización SIP es la correcta y esperada, y también que se produce de forma correcta la transparencia de los datos, tal y como se ha visto en el subapartado anterior. Se prueba con el softphone myMonster y también con IMSDroid, aunque éste último no soporta el envío de mensajes REFER, por lo que no se puede utilizar para iniciar el proceso de transferencia. Por último se crean escenarios en SIPp para así poder establecer llamadas simultáneas y probar a la vez distintos escenarios, obteniendo nuevamente la limitación de máximo una llamada por segundo que da el Core IMS. Los escenarios aparecen en B.4.1.5.

Habiéndose ya validado por separado las transferencias de tipo *Pull* y de tipo *Push*, se realizan también una serie de pruebas en las que se combinan ambos tipos transferencias, utilizando para ello combinaciones de los escenarios SIPp anteriormente creados. Se valida el resultado de estas pruebas verificando la señalización y el flujo de datos.

7.6 Medida del tiempo de transferencia

Finalmente se realiza una medida de cuánto es el tiempo que se tarda en completar de forma satisfactoria cada una de las transferencias, para así medir en cierta forma cuál es el rendimiento de la aplicación.

Se utilizaron los siguientes supuestos y condiciones para la toma de las medidas:

- Se utiliza movilidad en origen. Es el usuario que inicia la llamada el que va a realizar las transferencias.
- Los escenarios SIPp que simulan al llamante, el que realiza las transferencias, no van a mandar ningún tráfico de datos.
- El tráfico de datos lo enviará únicamente el llamado, que al establecer la sesión reproducirá un flujo RTP.

Para las transferencias de tipo *Pull*, se mide el tiempo desde que se envía el INVITE con la cabecera Replaces desde el nuevo terminal, hasta que llega el primer paquete de datos a dicho nuevo terminal. Se repite 30 veces la medida, obteniéndose un valor medio de 58 milisegundos.

Para las transferencias de tipo *Push*, se mide el tiempo desde que se envía el mensaje REFER desde el antiguo terminal, hasta que llega el primer paquete de datos al nuevo terminal. El tiempo variaría según lo que tardase en contestar el nuevo terminal, pero se configura el escenario SIPp para contestar con un 200 OK inmediatamente después de recibir el INVITE. Tras repetir la medida 30 veces, se obtiene un valor medio de 73 milisegundos.

El valor para las transferencias de tipo *Push* es mayor, lo cual se puede considerar como lógico al ser un proceso más complejo y que conlleva una mayor señalización. Los valores obtenidos son muy bajos, lo cual da una sensación de servicio prácticamente instantáneo a la hora de realizar transferencias, lo que implica una gran usabilidad de cara al usuario.

Capítulo 8

Conclusiones y trabajos futuros

A pesar de que el proyecto abarca muchas tecnologías y arquitecturas, todas extensas y de relativa complejidad, se puede concluir que el resultado general es plenamente satisfactorio, ya que se consigue cumplir con los diferentes subobjetivos y llegar al objetivo final de desarrollar una solución completa y poderla validar. A continuación se repasan los subobjetivos:

- Se utiliza la implementación de código abierto Open IMS Core de Fokus como plataforma IMS. Se consigue configurarla para poder desplegar el servicio de movilidad, y probarlo mediante usuarios.

- Se desarrolla la aplicación JAIN SLEE para el Mobility Manager, que actuando como un B2BUA de SIP consigue manejar toda la señalización SIP necesaria para establecer, liberar y transferir sesiones. Se despliega y se prueba de forma correcta en la implementación de código abierto Mobicents.

- Se desarrolla el Transparency Manager como una aplicación Java stand-alone, que se controla desde el Mobility Manager usando RMI. En este punto fue necesario un cambio en la idea inicial que se tenía para el desarrollo, como se ha visto en 6.2.

- Utilizando SIPp se consigue simular terminales para poder probar todos los casos documentados. También se consigue probar la solución utilizando softphones de IMS desarrollados por terceros, lo que da una mayor visibilidad y un valor añadido a la solución.

- Se realiza una medida básica del rendimiento en términos de llamadas simultáneas soportadas, y tiempos de transferencia. Se concluye que el Mobility Manager soporta sin problemas decenas de llamadas por segundo, mientras que la máquina virtual con el Open IMS Core que se ha utilizado limita esta tasa a una única llamada por segundo. Así mismo, los dos tipos de transferencia se muestran como prácticamente instantáneos de cara al usuario, al necesitar para su procesamiento tiempos inferiores a la décima de segundo.

- La solución se desarrolla de forma modular, y ajustada en todo momento a los estándares. No se prueba con otra implementación de Core IMS al no disponer de ninguna otra de código abierto, ni se prueba la aplicación JAIN SLEE en otro contenedor distinto al de Mobicents por el mismo motivo.

Aun cumpliendo el objetivo principal, hay aspectos del diseño y del desarrollo que podrían ser mejorados.

- El utilizar la máquina virtual ya configurada para el Open IMS Core permite tener un entorno operativo de manera más rápida, pero a costa de que este entorno no esté muy actualizado, ni ofrezca el mismo rendimiento que podría dar el desplegar la solución desde cero en una máquina independiente.

- Como se comentó en 4.2.1, comunicar el Mobility Manager con el Transparency Manager utilizando RMI no es la mejor forma de hacerlo desde el punto de vista de JAIN SLEE, habiendo sido la mejor solución el comunicarlos mediante un RA.

- El Transparency Manager tiene una gran sobrecarga de clases y de estado, heredado de la idea inicial de usar click. Lo ideal hubiera sido, una vez decidido el cambio de diseño, intentar adaptarlo a una implementación algo más sencilla.

- El desarrollo del Transparency Manager utilizando sockets, que se bloquean a la espera de datos y se cierran lanzando excepciones, podría mejorarse utilizando canales no bloqueantes u otras librerías Java de media. Así mismo, se podría probar a utilizar directamente algún tipo de Media Server.

- Se han probado detalladamente los distintos casos de uso, pero desde un punto de vista muy funcional, podrían haberse realizado pruebas de rendimiento más completas y exhaustivas, intentando determinar de una forma más concreta el rendimiento.

El trabajo de este proyecto puede ser continuado en muchos aspectos, considerándose en primer lugar el mejorar alguno de los aspectos arriba señalados. Además de esto, otra serie de posibles mejoras y extensiones serían:

- Permitir desde el Mobility Manager la renegociación de sesiones, procesando mensajes re-INVITE o UPDATE.

- Continuando con el Mobility Manager, permitir distintas formas de negociar inicialmente la sesión, tal y como se ha comentado en 4.2.2.

- Desarrollar el poder procesar transferencias de tipo *Push* en las que la petición REFER vaya en un nuevo diálogo.

- Montar una mejor estructura para las transferencias de tipo *Pull*, de forma que se guarde la información en alguna fuente externa como pueda ser una base de datos, y que esté organizada para que los distintos usuarios puedan consultar la información de las sesiones que tienen activas.

- Afrontar el desarrollo de un nuevo terminal, o modificar uno existente, para poder probar las transferencias de tipo *Pull*, y al mismo tiempo interactuar con el posible desarrollo del punto anterior.

El proyecto se ha enfocado de una forma genérica, para dar movilidad a usuarios independientemente de quién sea el destino con el que tenga establecida la sesión, que podría soportar o no procedimientos de transferencia. Sin embargo, tiene una posible aplicación muy concreta en el caso de IPTV.

En la arquitectura de IPTV para IMS, el servidor de contenidos no va a soportar por sí mismo procedimientos de transferencia, pero sí que es un servicio que sería muy bien aceptado por el usuario el poder transferir su sesión de IPTV entre sus distintos terminales, por ejemplo para continuar viendo la televisión en el móvil una vez que se abandona el hogar [30].

Apéndice A

Manual de instalación

En este primer apéndice del proyecto se detallarán los pasos necesarios para poder instalar todos los componentes que conforman la solución. Se dividirá en distintos subapartados, uno para cada uno de dichos componentes. Todo lo necesario para la instalación se proporciona en el CD que se entrega junto a esta memoria.

A.1 Core IMS

Para poder instalar el Core IMS es necesario disponer del software VMware Player, que servirá para poder arrancar la máquina virtual que lo contiene. Para el desarrollo del proyecto la máquina host que contiene a la máquina virtual del Core IMS es una máquina Linux, con el sistema operativo Ubuntu 10.04 LTS de 32 bits; las instrucciones que se detallarán a continuación serán para la instalación en una máquina similar, con sistema operativo Linux de 32 bits. Es posible instalar el programa en una máquina con distinto sistema operativo, para ello habrá que obtener el software apropiado y seguir las instalaciones que se detallan en la web del programa [27].

En el directorio "Core IMS/VMware Player/" del CD adjunto se encuentra el ejecutable del programa junto a un PDF con instrucciones más detalladas para su manejo. Para instalar el programa se copiará el ejecutable a la máquina donde se quiera instalar, y abriendo un terminal en el directorio en que se haya copiado se ejecutará el siguiente comando, siendo necesario tener permisos de superusuario:

```
$ sudo sh VMWare-Player-4.0.3-703057.i368.txt
```

Con esto ya se tendría instalado el programa, que aparecerá en el listado de aplicaciones. Se podrá ejecutar seleccionándolo desde dicho listado de aplicaciones, o ejecutando en un terminal el comando `$vmplayer`.

El siguiente paso consistirá en copiar a la máquina de instalación los archivos de la máquina virtual, que se encuentran en el directorio "Core IMS/OpenIMSCore.Ubuntu". Una vez hecho esto se arrancará la máquina virtual utilizando VMware Player, seleccionando en el programa la opción de abrir una máquina virtual. La primera vez que se arranque la máquina virtual se escogerá la opción que indica que la máquina se ha copiado, y para posteriores arranques la máquina aparecerá en el listado nada más abrir la aplicación VMware Player.

El usuario de la máquina virtual es `root`, y su contraseña es `password`. Al arrancar la máquina ya se encuentran funcionando todos los componentes del Core IMS, pero la dirección IP configurada es 192.168.0.193. Es necesario cambiar dicha dirección IP por la que tenga asignada la máquina virtual, que está configurada de tal modo que replique la conexión física

de la máquina host, y obtenga una nueva dirección IP del mismo modo en que el sistema host la obtuvo. Una vez conocida la dirección IP son necesarios los siguientes cambios:

- En el directorio "/opt/OpenIMSCore/etc/bind" se modifica el fichero `db.open-ims.test`, sustituyendo las ocurrencias de la antigua dirección IP por la nueva dirección IP de la máquina. Esto es necesario para que el DNS de la máquina virtual resuelva de forma correcta la dirección que corresponde a cada uno de los componentes del Core IMS. Una vez modificado el fichero se reinicia el servidor DNS, ejecutando el siguiente comando: `$ /etc/init.d/bind9 restart`. Para comprobar que el DNS está funcionando de forma correcta se puede probar a hacer ping a open-ims.test, y verificar que se responde al ping con la nueva dirección IP.
- Si se quiere resolver el resto de dominios aparte del correspondiente al propio Core se puede editar también el fichero `named.conf.options` del mismo directorio, sustituyendo en el apartado `forwarders` las direcciones IP que aparecen por aquellas correspondientes a los servidores DNS que se utilicen en la red. Una vez modificado el fichero es necesario reiniciar nuevamente el servidor de DNS tal y como vio en el punto anterior.
- Se actualiza también la dirección IP en los siguientes ficheros de configuración. En el directorio "/opt/OpenIMSCore/etc/" los ficheros `pcscf.cfg`, `scscf.cfg`, `scscf.xml`, `icscf.cfg` y `icscf.xml`. En el directorio "/opt/OpenIMSCore/etc/hss" los ficheros `hss.properties` y `DiameterPeerHSS.xml`.
- Tras realizar todos los cambios será necesario reiniciar los componentes del Core, *CSCFs* y *HSS*. Para ello abriremos un terminal en el directorio "/opt/OpenIMSCore/bin" y ejecutaremos los siguientes comandos:
  `$ ./kill.sh`, que parará todos los componentes.
  `$ ./start.sh`, que los iniciará.
- El último cambio necesario consiste en indicar cuál será la dirección IP de la máquina donde esté instalado el Mobility Manager, para que se le redirija la señalización SIP en los casos en que sea necesario. Para ello habrá que acceder a la web de configuración del HSS y modificar dicha dirección. La configuración web del HSS se explica más detalladamente en B.1, se resumen aquí los pasos necesarios para únicamente actualizar esta dirección:
  - Se abre un navegador web, y se accede a la dirección http://IP:8080, donde IP será la dirección IP de la máquina virtual.
  - El usuario para acceder a la web de configuración es `hssAdmin`, y su contraseña `hss`.
  - Se accede a la pestaña superior de servicios, y a continuación se escoge la opción buscar del apartado de servidores de aplicación de la izquierda.
  - Se pulsa el botón de buscar y se selecciona el Mobility Manager. A continuación se modifica la dirección IP y se pulsa el botón guardar para que se guarde la modificación. El resultado debe ser similar al que aparece en la Figura 20.

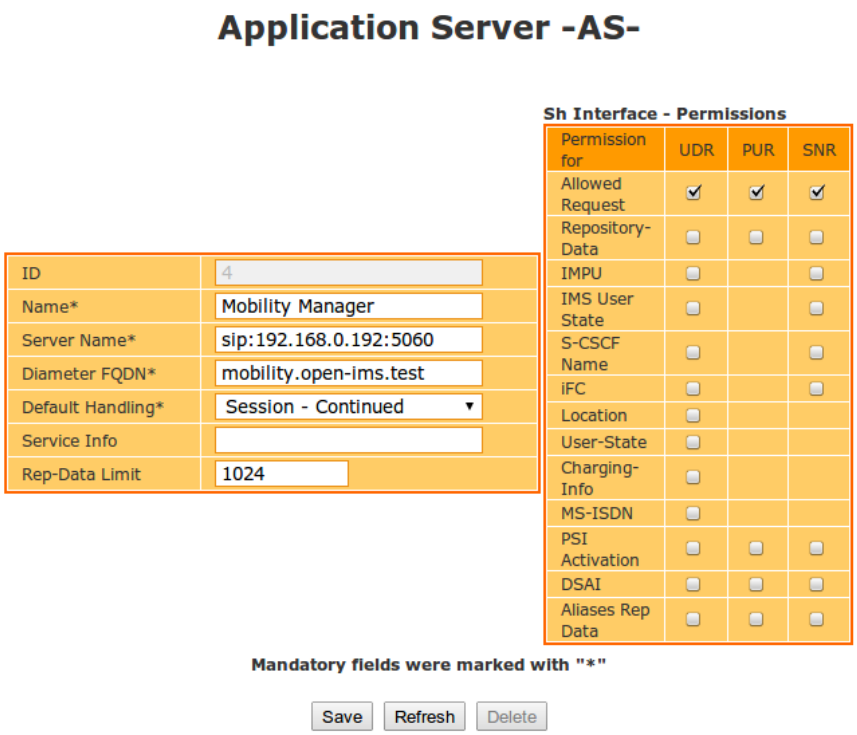

Figura 20: Configuración de la dirección IP del Mobility Manager

A.2 Mobility Manager

Para poder poner en funcionamiento el Mobility Manager se necesitan como requisitos en la máquina en que se vaya a desplegar tener instalada una versión del JDK 6 de Java, y definir una serie de variables de entorno. Se detallan todos los pasos para desplegar en una máquina con sistema operativo Ubuntu 10.04 LTS de 32 bits, si se quisiera instalar en otro sistema habría que realizar los pasos equivalentes para dicho sistema operativo.

En el directorio "JDK" del CD adjunto se encuentra el ejecutable para instalar el jdk, jdk-6u32-linux-i586.bin. Se copia dicho ejecutable a la carpeta del sistema donde si quiera instalar, por ejemplo "/home/isma/tools". A continuación se ejecutan los siguientes comandos en un terminal:

- `$ chmod a+x jdk-6u32-linux-i586.bin`, para dar permisos de ejecución.
- `$ ./jdk-6u32-linux-i586.bin`, para instalar.

El contenedor JAIN SLEE de Mobicents donde se desplegará el servicio del Mobility Manager se encuentra en la carpeta mobicents-jainslee, dentro del directorio "Mobility Manager" del CD adjunto. Copiamos dicha carpeta a la máquina de instalación, por ejemplo en "/home/isma". Siguiendo la estructura de directorios usada como ejemplo, en "/home/isma/mobicents-jainslee/jboss-5.1.0.GA/server/default/deploy" encontramos los siguientes archivos:

- `sip11-ra-DU-2.6.0.FINAL.jar`. Es la unidad desplegable que contiene el RA de SIP a desplegar.
- `mobility-manager-DU.jar`. Es la unidad desplegable que contiene el servicio JAIN SLEE del Mobility Manager, junto con los SBBs y la librería que lo componen.
- `properties-service.xml`. Aquí se definen las properties para el servicio, será necesario editar los valores acorde al entorno:
    - `rmiRegistryHost`: Se configura con la dirección IP de la máquina donde se registre por RMI el Transparency Manager.
    - `rmiRegistryPort:` El puerto para RMI.
    - `pullTransferFilesPath:` Directorio donde guardar los ficheros con la información necesaria para realizar transferencias de tipo *Pull*, como se vio en 7.4.
    - `expiresValue:` Valor en segundos para la suscripción que supone el mensaje REFER en transferencias de tipo *Push*.
    - `pushTransfertimeout:` Tiempo en segundos que se espera para poder realizar una transferencia de tipo *Push*. Como se vio en 5.2.2, su valor debe ser inferior al valor de expires, para que así no sea necesario refrescar la suscripción.
    - `origSCSCF`: SCSCF al que enviar las peticiones INVITE para transferencias de tipo *Push*, el valor que tiene sirve para el entorno Open IMS Core utilizado.

A continuación es necesario definir una serie de variables de entorno para que el contenedor Mobicents funcione de forma correcta. Para ello se edita, o en caso de que no exista se crea, el archivo .pam_environment [28] en el directorio hogar del usuario, en el caso de ejemplo "/home/isma". Se definen las variables `JAVA_HOME` y `JBOSS_HOME`, y se añaden los ejecutables del JDK al PATH del sistema. La Figura 21 muestra el contenido del fichero para la estructura ejemplo de directorios que se está usando:

```
JAVA_HOME DEFAULT=/home/isma/tools/jdk1.6.0_32
JBOSS_HOME DEFAULT=/home/isma/mobicents-jainslee/jboss-5.1.0.GA
PATH DEFAULT=${PATH}:${JAVA_HOME}/bin
```

Figura 21: Fichero .pam_environment

Para que el Mobility Manager funcione correctamente es necesario que, en la máquina en que se instale, se utilice como DNS la IP de la máquina virtual que contiene el Core IMS. Esto se debe a que el Mobility Manager necesita resolver el dominio open-ims.test, y más concretamente la dirección scscf.open-ims.test, para así poder enrutar los mensajes SIP de forma correcta a través del Core IMS.

Con estos pasos ya se tendría todo correctamente configurado para poder desplegar el Mobility Manager. Los pasos necesarios para arrancar el servidor, ver si todo está correctamente desplegado, y parar el servidor, aparecen en B.2.

A.3 Transparency Manager

El Transparency Manager consiste en una aplicación Java stand-alone que se proporciona comprimida en un fichero JAR. Para poder ejecutar la aplicación se necesita disponer de un JRE o un JDK de Java instalado, y cuyos ejecutables se encuentren en el PATH del sistema. Los pasos para instalar un JDK y añadir sus ejecutables al PATH del sistema se detallan en el subapartado anterior, A.2.

El Transparency Manager puede ejecutarse en la misma máquina que el Mobility Manager o en otra independiente, al desarrollarse como entidades independientes que permiten separar el camino de señalización del camino de los datos, tal y como ya se ha explicado.

El archivo JAR que contiene la aplicación se llama `transparencyManager.jar`, y se encuentra en el directorio "Transparency Manager" del CD adjunto. Se copiará este archivo a un directorio de la máquina en que se quiera desplegar. La configuración necesaria para poder ejecutar consiste en editar el fichero que contiene las properties para la apliación, dicho fichero se denomina `transparency.properties`, y se encuentra en la carpeta "resources" del archivo JAR. Las properties son las siguientes:

- `initialPort:` Puerto inicial para la serie de puertos que utilizará el Transparency Manager.
- `numPorts:` Número de puertos que se utilizarán, y cuyos valores parten del de `initialPort`.
- `transparencyManagerIP:` Se configurará con el valor de la IP donde se desplegará el Transparency Manager, y que será la IP por la que se redirijan los flujos de datos.
- `bufferSize:` Tamaño máximo para los paquetes que se esperan recibir.

Los procedimientos para arrancar y parar el Transparency Manager se encuentran en B.3.

A.4 Terminales

En este último subapartado del manual de instalación se indicará la forma de instalar los terminales utilizados para validar la solución: SIPp, y los softphones myMonster e IMSDroid.

A.4.1 SIPp

SIPp se va a utilizar para programar escenarios que se registrarán en el Core IMS y que enviarán tráfico RTP, tal y como se ha visto en el Capítulo 7. Debido a esto para instalarlo es necesario compilar el código fuente para que se soporten estas dos funcionalidades.

El código fuente se encuentra en la carpeta "SIPp", dentro del directorio "Terminales" del CD adjunto. Se copia esta carpeta a la máquina desde donde se quieran ejecutar los scripts de

SIPp. Para compilar en un sistema Linux con Ubuntu 10.04 de 32 bits, se necesitan instalar los siguientes paquetes, mediante el uso del comando `$ sudo apt-get install`:

- `openssl`
- `openssl-blacklist`
- `openssl-blacklist-extra`
- `libssl-dev`
- `libpcap-dev`
- `libncurses-dev`

Una vez instalados estos paquetes se procede a compilar, para lo cual se abrirá un terminal en el directorio donde se haya copiado el código fuente, y se ejecutará el comando `$ make pcapplay_ossl`. Si se compila de forma correcta aparecerá en el directorio un ejecutable de nombre `sipp`. Si surgen problemas durante la instalación, o se quiere compilar para otro tipo de sistema, se puede consultar en la página web de la herramienta [20].

Se puede exportar al PATH del sistema el directorio donde se encuentra el ejecutable, para así no necesitar facilitar el path completo a la hora de ejecutar escenarios. Para ello se puede editar el fichero .pam_environment que aparece en la Figura 21, añadiendo el directorio correspondiente al valor de la variable PATH.

El procedimiento para ejecutar escenarios, así como los escenarios usados para probar la solución aparecen en B.4.1.

A.4.2 MyMonster

En el directorio "Terminales" del CD adjunto se incluye una carpeta "MyMonster" que contiene la versión del terminal para Linux, que es el entorno en el que se ha probado. Para poder ejecutar el terminal se necesita disponer de una versión 6 del JRE o el JDK de Java, los pasos para instalar el JDK de Java y exportar los ejecutables al PATH del sistema se pueden encontrar en A.2.

Para poder reproducir correctamente los flujos RTP y mostrar el audio y/o vídeo recibidos, y a su vez capturar vídeo de la webcam y audio del micrófono (en caso de que se tengan estos dispositivos en la máquina de instalación), es necesario instalar una serie de paquetes en el sistema. Para un sistema Ubuntu 10.04 LTS de 32 bits, es necesario instalar mediante el comando `$ sudo apt-get install` los siguientes:

- `libgstreamer0.10-dev`
- `libgstreamer-plugins-base0.10-dev`
- `gstreamer0.10-plugins-bad`
- `gstreamer0.10-plugins-bad-multiverse`
- `gstreamer0.10-plugins-base`
- `gstreamer0.10-plugins-farsight`
- `gstreamer0.10-plugins-good`

- `gstreamer0.10-plugins-bad`
- `gstreamer0.10-plugins-bad-multiverse`
- `gstreamer0.10-ffmpeg`

Los procedimientos necesarios para configurar y usar este terminal aparecen en B.4.2.

A.4.3 IMSDroid

El *softphone* IMSDroid para el sistema operativo móvil Android es un software en evolución cuyo objetivo en el proyecto consiste en dar una mayor visibilidad y un valor añadido a la solución implementada, comprobando que es posible el establecer sesiones con dispositivos móviles.

Como ya se vio en el Capítulo 7 es un software que no permite realizar transferencias, ni siquiera las de tipo *Push* como es el caso de myMonster, ya que no soporta el envío de REFER; aun así se puede utilizar para iniciar y recibir llamadas, así como para ser el objetivo de transferencias de tipo *Push*, o para ser el terminal desde el que obtener la sesión en transferencias de tipo *Pull*.

Para poder instalar y ejecutar el software se necesita disponer de un dispositivo móvil con sistema operativo Android. En el momento en el que se finaliza la realización de este proyecto la aplicación no se encuentra disponible en la Play Store de Android, por lo que es necesaria su instalación de forma manual. En el directorio "Terminales/IMSDroid" del CD adjunto se encuentra la versión que se ha utilizado en el proyecto, que consiste en un archivo APK listo para instalarse en un terminal Android; dicho archivo se obtuvo directamente de la web de la aplicación [22].

Los pasos necesarios para instalar en el terminal Android serían:

- En caso de que no esté hecho ya, modificar los ajustes de seguridad del dispositivo para permitir instalar aplicaciones desde orígenes desconocidos.
- Descargar en el terminal Android el archivo APK del CD de instalación.
- Abrir el archivo desde el terminal y aceptar los permisos, con lo que quedará instalada la aplicación.

Los procedimientos necesarios para configurar y usar la aplicación se encuentran en B.4.3.

Apéndice B

Manual de usuario

Este apéndice del proyecto complementa al anterior, y especifica todos los procedimientos necesarios para manejar cada uno de los componentes, y para poder probar la solución desarrollada. Se subdivide también en distintos subapartados, uno para cada componente de la solución.

B.1 Core IMS

Este subapartado se dividirá para tratar individualmente cada uno de los aspectos en que se trabaja con el Core IMS: como arrancar y parar sus componentes, como dar de alta servicios, y cómo dar de alta usuarios.

B.1.1 Arranque y parada

Para arrancar la máquina virtual que alberga todos los componentes del Core IMS se procederá como se ha visto en A.1. Cada vez que se arranque la máquina será necesario un reinicio de los componentes usando los comandos `kill` y `start`, como ya se comentó también.

Una vez arrancados los componentes, se puede acceder a los logs de cada uno de ellos para comprobar que estén funcionando de manera correcta, ver los registros de usuarios, establecimientos de sesiones,… Para ello habrá que arrancar un terminal en el directorio "/opt/OpenIMSCore/bin", y desde ahí ejecutar los siguientes scripts:

- `./attach.fhoss.sh`: Para acceder a los logs del *HSS*.
- `./attach.pcscf.sh`: Para acceder a los logs del *P-CSCF*.
- `./attach.scscf.sh`: Para acceder a los logs del *S-CSCF*.
- `./attach.icscf.sh`: Para acceder a los logs del *I-CSCF*.

Con la ejecución de los scripts anteriores se accede a los logs de los componentes mediante el emulador de terminales screen [29], al ser la forma en que está configurado en la máquina virtual proporcionada por el instituto FOKUS. Los comandos más importantes para poder manejarse en el entorno serían:

- Pulsar la combinación de teclas `crtl-a`, y posteriormente `esc`. Así se entraría en modo copia, y se podría hacer scroll para consultar más en detalles los logs. Pulsando nuevamente `esc` se saldría de este modo.

- Para salir del log, se pulsaría la combinación de teclas `ctrl-a`, y posteriormente d. No hay que pulsar la combinación `crtl-c`, pues esto supondría parar el componente.

Para poder capturar y verificar el tráfico de señalización SIP se puede arrancar el programa Wireshark que viene ya instalado en la máquina virtual. Los mensajes intercambiados entre los distintos componentes del Core aparecerían capturando en la interfaz de loopback, mientras que los mensajes intercambiados con componentes que estén en otras máquinas (terminales, Mobility Manager) se verían capturando en la interfaz de red; capturando en la pseudo-interfaz any se verían todos. El puerto en que recibe la señalización SIP el *P-CSCF* es el 4060, el *I-CSCF* el 5060, y el *S-CSCF* el 6060.

El modo de acceder la interfaz web de configuración del *HSS* también se describió en A.1, será necesario abrir en un navegador web la dirección http://IP:8080, y utilizar el usuario/contraseña `hssAdmin/hss`. Se puede acceder a la web desde la misma máquina virtual, o desde cualquier otra máquina desde la que se tenga acceso.

B.1.2 Servicio de movilidad de sesiones

La máquina virtual del Core IMS se proporciona con el servicio de movilidad de sesiones ya correctamente configurado. Lo necesario para proveer dicho servicio se especificó en 3.1, en este subapartado se mostrarán imágenes de cómo es la configuración vista directamente en la interfaz web del *HSS*.

- Pulsando en la pestaña superior servicios, aparecen a la izquierda los accesos necesarios para poder configurar un servicio. El primer punto a configurar es el *Application Server*, dando de alta el Mobility Manager. La configuración es la misma que aparece en la Figura 20, y que se vuelve a mostrar en la Figura 22.

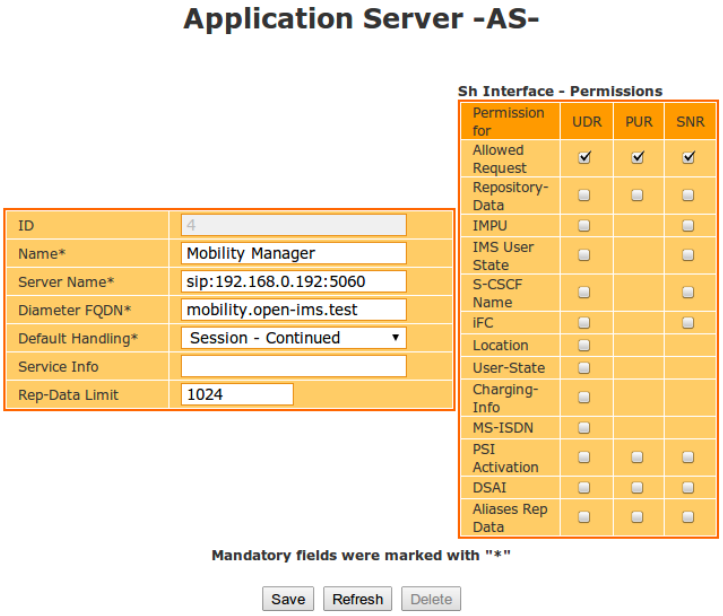

Figura 22: Configuración del Mobility Manager como *Application Server*

- A continuación se configura el *Trigger Point*, que permita redirigir la señalización SIP al Mobility Manager en los casos que ya han sido especificados. La configuración se observa en la Figura 23.

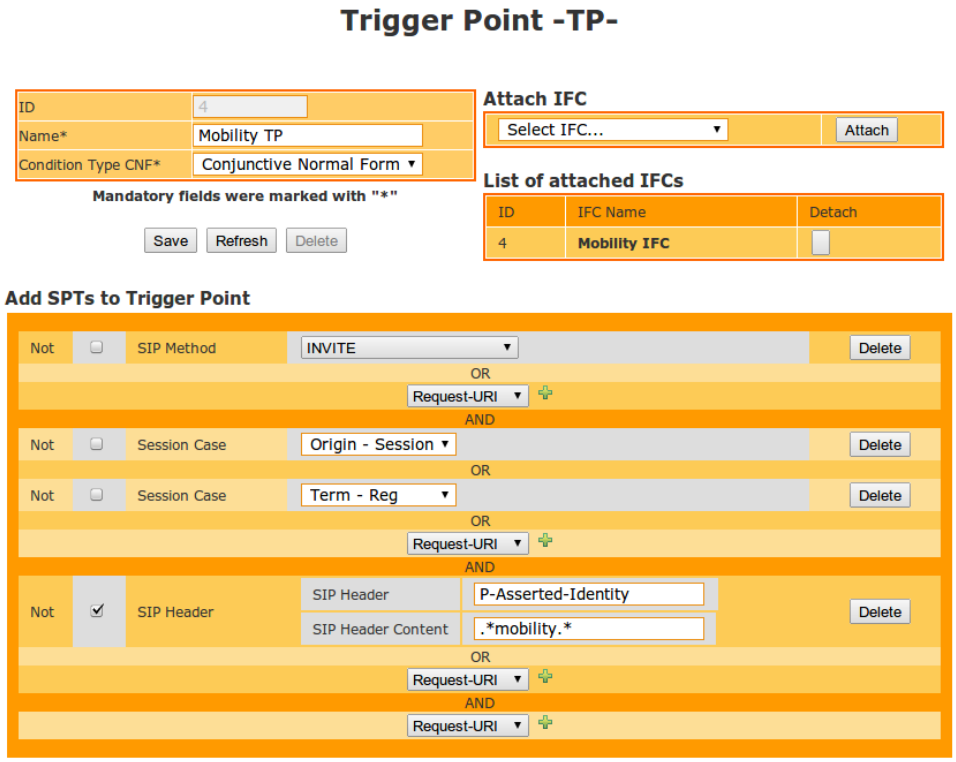

Figura 23: Configuración del *Trigger Point*

- El siguiente paso es asociar en un nuevo *Initial Filter Criteria* el *Application Server* y el *Trigger Point* anteriores. Se puede ver la configuración en la Figura 24.

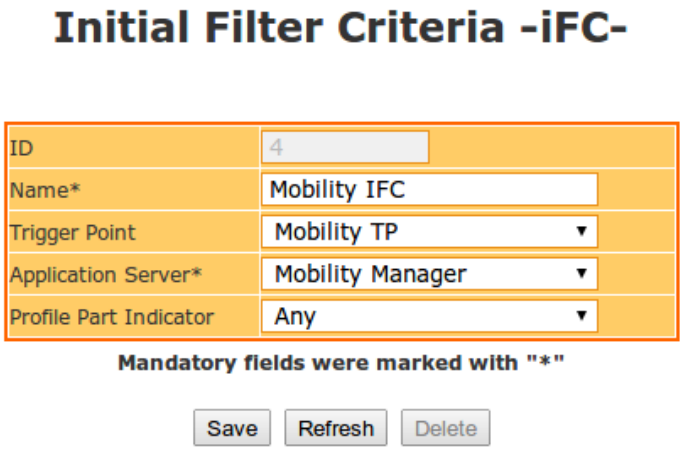

Figura 24: Configuración del *Initial Filter Criteria*

- Ya el último paso consiste en crear un nuevo *Service Profile* y asociar el *Initial Filter Criteria* anterior. Este service profile es el que se asignará a las identidades públicas de los usuarios con el servicio de movilidad. La configuración del *Service Profile* aparece en la Figura 25.

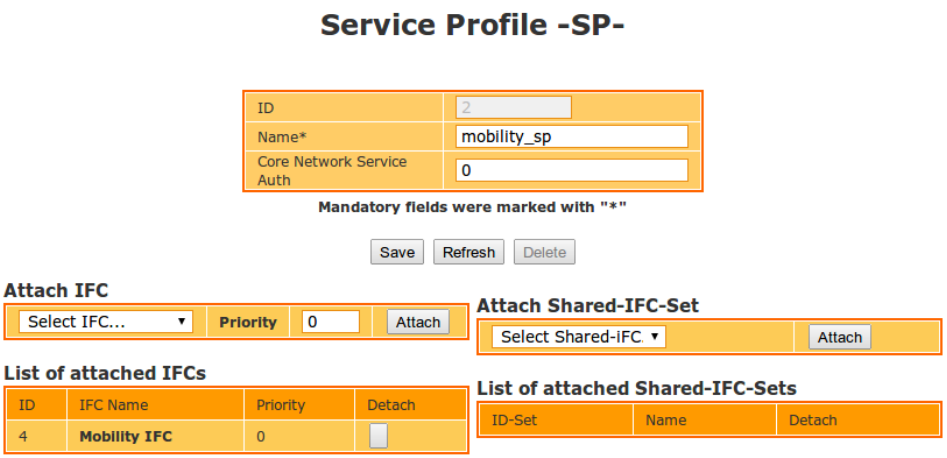

Figura 25: Configuración del *Service Profile*

Como también se especificó en 3.1, es necesario dar de alta un PSI que identifique al Mobility Manager, y que será necesario para realizar las transferencias de tipo *Push*. Los pasos para dar de alta un PSI difieren poco de los necesarios para dar de alta un usuario, y que se verán en el siguiente subapartado. Es necesario en este caso ir a la pestaña superior de usuarios, se recoge a continuación la configuración necesaria:

- Se da de alta una nueva suscripción en el Core IMS, una nueva suscripción o IMSU. La configuración se observa en la Figura 26.

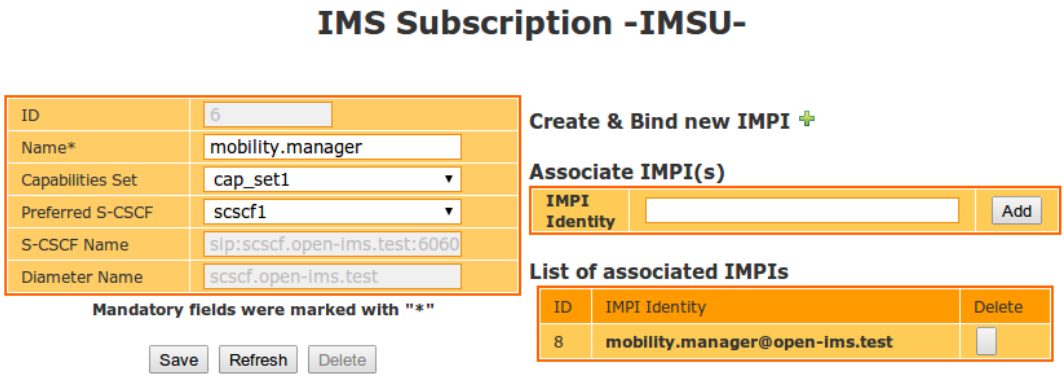

Figura 26: Configuración de IMSU para PSI

- Se crea una nueva identidad privada o IMPU para la suscripción anterior. Son indiferentes los procedimientos que pongamos para el registro, pues la identidad no se registrará. La Figura 27 recoge la configuración.

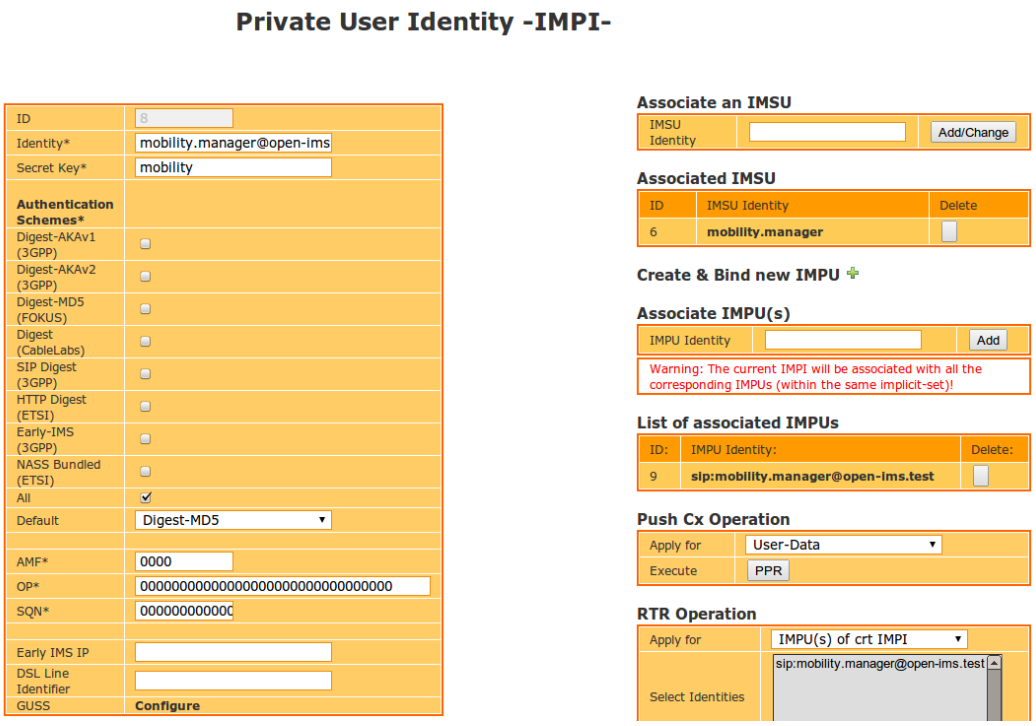

Figura 27: Configuración de IMPI para PSI

- Finalmente se crea el PSI. En esta configuración radica la diferencia con dar de alta un usuario, pues se escoge la opción de PSI en lugar de la de *Public Identity*. Se asocia el perfil por defecto sin servicios asociados ya que no se le quiere dar ningún tratamiento a las llamadas originadas o dirigidas al PSI, el tratamiento es el servicio de movilidad para los usuarios. La configuración es la que aparece en la Figura 28.

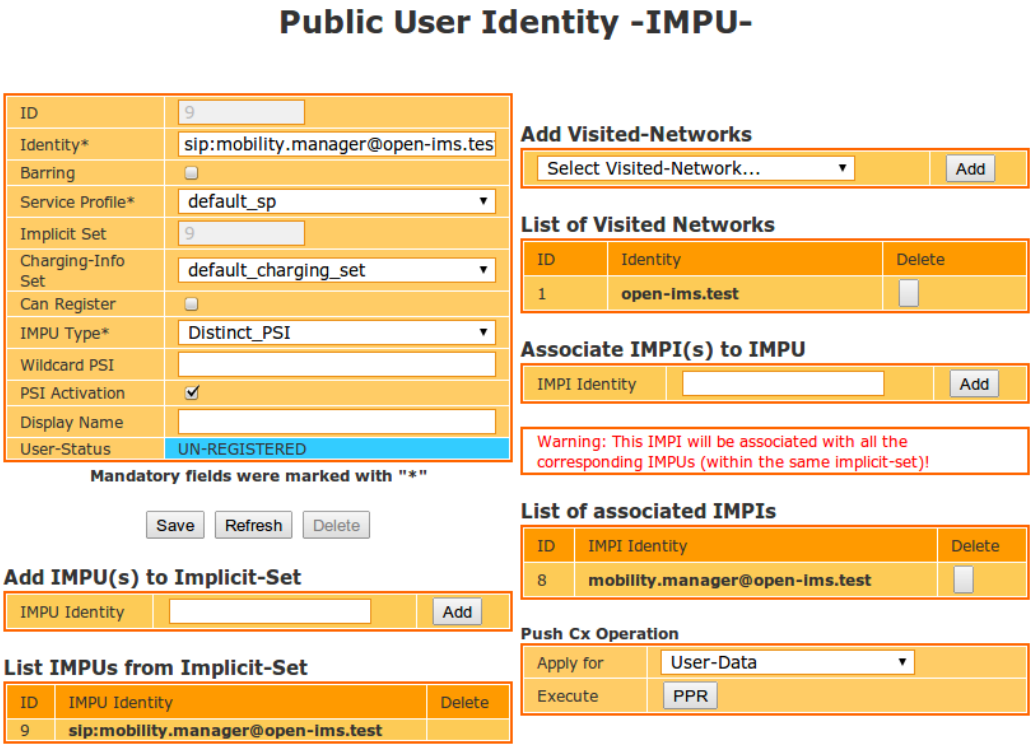

Figura 28: Configuración de IMPU para PSI

B.1.3 Usuarios

Al igual que ocurría con el servicio de movilidad, la máquina virtual se entrega con una serie de usuarios ya configurados y que permiten probar todos los posibles escenarios. Los usuarios configurados son los siguientes:

- ***isma***. Es un usuario con movilidad. Su identidad privada es isma@open-ims.test y tiene tres identidades públicas distintas, todas con el *Service Profile* de movilidad:
  - sip:isma@open-ims.test
  - sip:isma.movil@open-ims.test
  - sip:isma.pc@open-ims.test
- ***noemi***. Es también un usuario con movilidad. En este caso su identidad privada es noemi@open-ims.test, y tiene también tres identidades públicas distintas con el *Service Profile* de movilidad:
  - sip:noemi@open-ims.test
  - sip:noemi.movil@open-ims.test
  - sip:noemi.fijo@open-ims.test
- ***carmen***: Este es un usuario que no dispone del servicio de movilidad. Su identidad privada es carmen@open-ims.test, y dispone de una única identidad pública: sip:carmen@open-ims.test.

Con estos tres usuarios se pueden probar los tres escenarios vistos en el Capítulo 7:

- Llamando desde una de las identidades públicas de isma o noemi a la identidad pública de carmen se tiene el caso definido como movilidad en origen.
- Llamando ahora desde la identidad pública de carmen a una de las identidades públicas de noemi o isma se tiene el caso de movilidad en destino.
- Finalmente llamando desde una de las identidades públicas de isma a una de las identidades públicas de noemi, o haciendo lo opuesto, se tiene el caso de la doble movilidad.

Se pueden crear nuevas identidades o modificar las existentes de forma sencilla. A continuación se detalle el proceso para dar de alta desde cero una nueva suscripción en el Core, asociada a una identidad privada y a una única identidad pública.

- El primer paso consiste en crear una nueva suscripción. Se selecciona en primer lugar la pestaña superior de usuarios dentro de la interfaz de gestión del *HSS*, y a continuación la opción de crear debajo de IMSU. Se escoge el nombre que se quiera para la suscripción, por ejemplo nasta, y se eligen las capacidades y el *S-CSCF* por defecto. El resultado puede verse en la Figura 29.

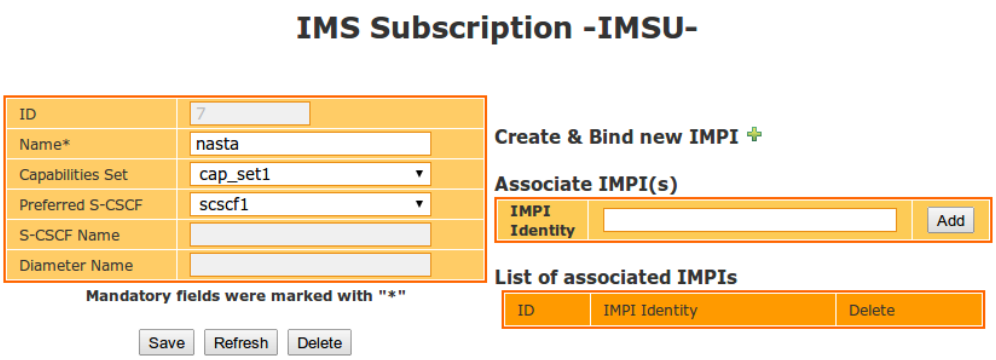

Figura 29: Creación de una nueva IMSU

- A continuación se pulsa el botón de crear y asociar una nueva identidad privada. Se escoge el nombre para la identidad, por ejemplo nasta@open-ims.test, y se le da un valor a la clave, por ejemplo nasta. Se eligen los algoritmos permitidos para el registro, pudiéndose escoger el número que se desee así como todos, y se escogerá en último lugar el algoritmo por defecto. Si se va a registrar la identidad utilizando SIPp será necesario escoger el algoritmo Digest-MD5, pues es el que se usa para los escenarios de registro. El resultado de la configuración puede verse en la Figura 30.

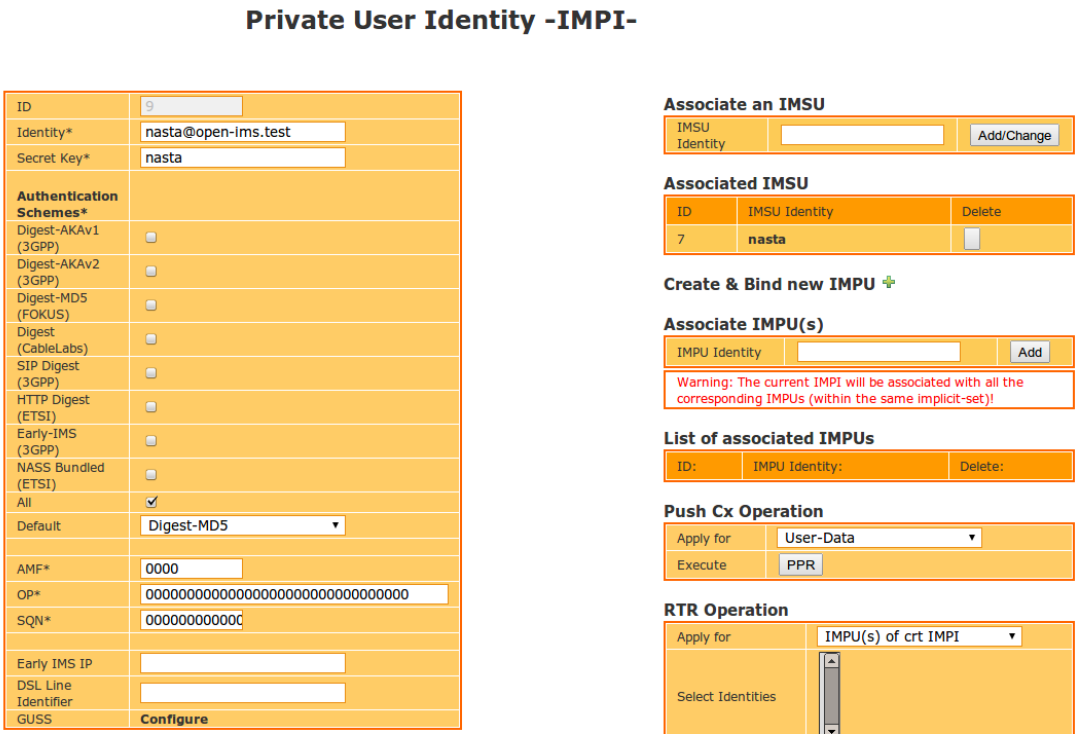

Figura 30: Creación de una nueva IMPI

- En último lugar se pulsará el botón de crear y asociar una nueva identidad pública. Se le da un valor a la identidad, por ejemplo sip:nasta@open-ims.test, se escoge el *Service Profile* teniendo en cuenta si se le va a dar el servicio de movilidad al usuario, se elije el valor por defecto para la información de cobro, se elije la opción para que se pueda registrar, y finalmente como tipo de identidad se escoge lo que se está configurando, identidad privada de usuario. Se añade como red visitada open-ims.test, para que el usuario pueda registrarse correctamente, y se pulsa el botón de guardar,

quedando totalmente configurado y dado de alta el nuevo usuario. La Figura 31 muestra la configuración explicada.

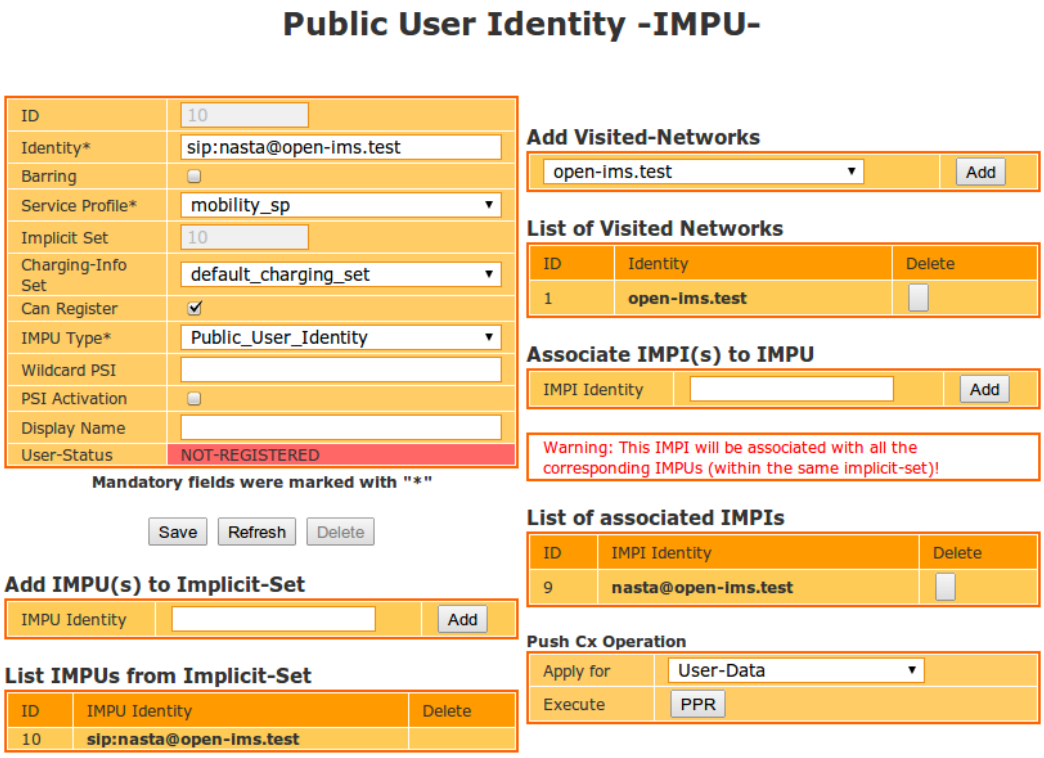

Figura 31: Creación de una nueva IMPU

B.2 Mobility Manager

Una vez instalado el contenedor JAIN SLEE de Mobicents y editadas las properties a las valores que se consideren para el entorno, tal y como se ha visto en A.2, se está en disposición de arrancar el servidor y desplegar la aplicación del Mobility Manager.

Para arrancar el servidor habrá que abrir un terminal en el directorio "$JBOSS_HOME/bin", y ahí ejecutar el siguiente comando: `$ ./run.sh -b IP`, donde IP es la dirección IP de la máquina en que se está arrancando. Si se quiere arrancar en *background*, habrá que añadir el símbolo & al final del comando.

Para parar el servidor bastará con pulsar la combinación de teclas `ctrl-c` en caso de haber arrancado en primer plano. Si se arrancó en *background*, o directamente para pararlo desde otro terminal, en el mismo directorio que en el caso anterior se ejecutará el comando `$ ./shutdown.sh -s IP -S`.

Si el servidor arranca de forma correcta no aparecerá ningún mensaje de error ni excepción en el terminal, y al final del arranque se verá una información similar a la que se observa en la Figura 32, donde se indica que el servidor ha arrancado y que el servicio del Mobility Manager se ha iniciado.

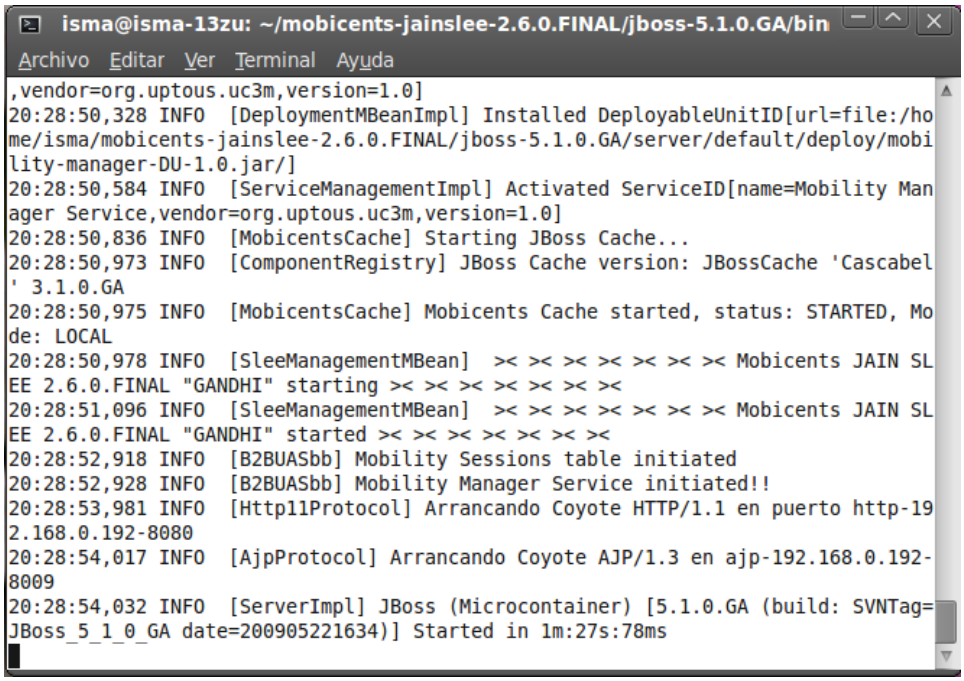

Figura 32: Arranque del Mobility Manager

Mobicents ofrece una interfaz web de gestión, donde se podrá observar que todos los componentes están correctamente desplegados. Para acceder a la interfaz habrá que abrir en un navegador la dirección http://IP:8080/slee-management-console, siendo IP nuevamente la dirección IP en que se ha arrancado el servidor. El aspecto de la web de gestión es el que aparece en la Figura 33.

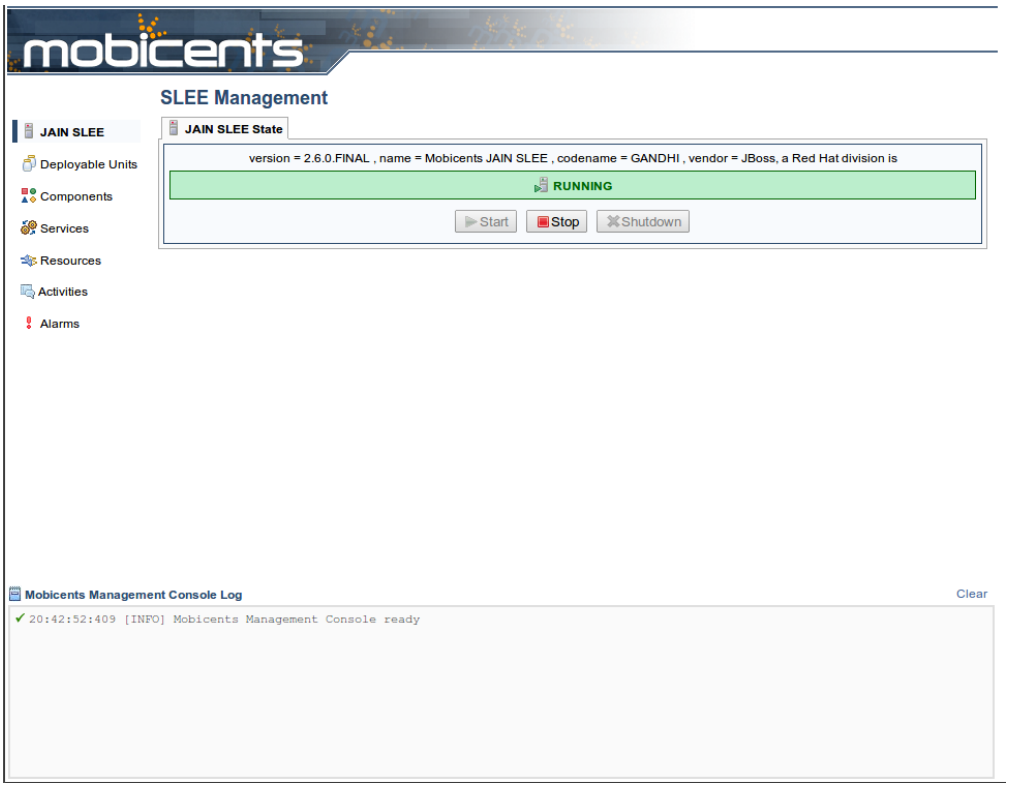

Figura 33: Mobicents Management Console

Navegando por el interfaz web se puede observar los componentes desplegados ya comentados a lo largo de la memoria, el servicio, los distintos SBBs, la librería y el *Resource Adaptor* de SIP. El puerto por defecto en que arranca el RA de SIP y espera recibir señalización es el 5060.

B.3 Transparency Manager

Una vez configurados correctamente los valores del fichero properties, tal y como se vio en A.3, se está en disposición de arrancar el Transparency Manager.

Lo primero que se necesita es arrancar el registro RMI, donde se exportará el objeto que utilizará el Mobility Manager para invocar la funcionalidad de transparencia. Para ello en un terminal del sistema se ejecutará el comando `$ rmiregistry port`. En `port` se especificará el puerto en que se quiere arrancar el registro, cuyo valor por defecto si no se indica ninguno es 1099. Si el Transparency Manager se va a desplegar en la misma máquina en que se tiene desplegado el Mobility Manager, el puerto por defecto ya está ocupado por el contenedor Mobicents, por lo que será necesario especificar otro.

A continuación se arrancará el propio servidor. Para ello se abrirá un terminal en el directorio en que se haya copiado el JAR del componente, y se ejecutará el siguiente comando: `$ java -jar transparencyManager.jar IP port`. Los argumentos IP y port hacen referencia a la IP de la máquina y al puerto donde se ha arrancado el registro RMI. El estado una vez arrancado el servidor será similar al que se puede apreciar en la Figura 34.

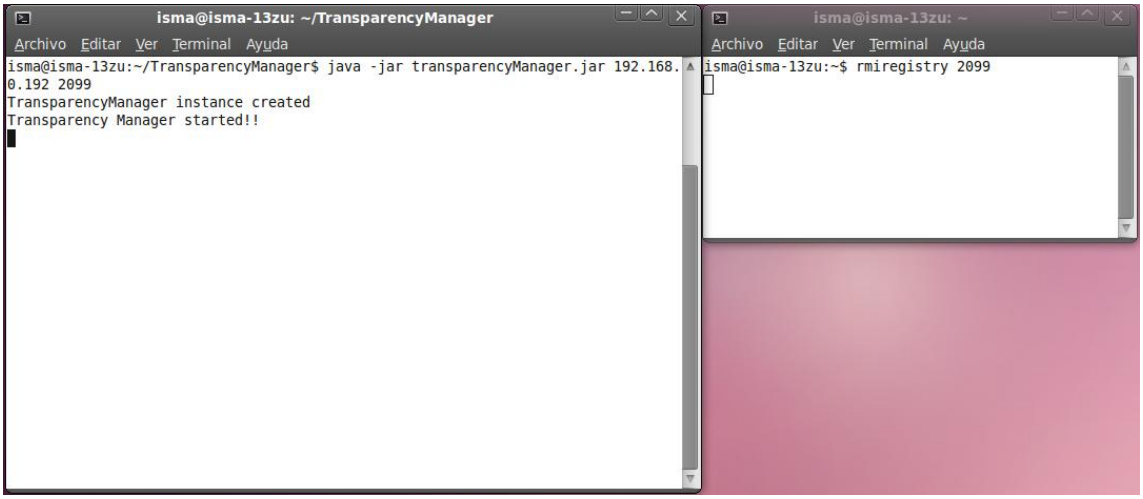

Figura 34: Arranque del Transparency Manager

Para parar el servidor, bastará con pulsar la combinación de teclas `crtl-c` en el terminal en que se ha arrancado.

B.4 Terminales

Este último subapartado del apéndice mostrará como configurar y usar los terminales utilizados para probar la aplicación. En el caso de SIPp se detallarán también los escenarios utilizados en la realización de las pruebas, y que son referenciados desde el Capítulo 7.

B.4.1 SIPp

La forma de ejecutar los scripts de SIPp es siempre la misma. Se explica en detalle a continuación, partiendo de la forma de invocarse y explicando cada uno de los posibles parámetros.

```
$ sipp –sf escenario direccionRemota –s destino –rsa ruta –auth_uri
dominio–i ipLocal –p puertoLocal –mp puertoLocalMedia –r tasa –rp
perido –m numLlamadas –inf fichero –rtp_echo
```

- `escenario:` El fichero xml con el escenario que se quiere ejecutar.
- `direccionRemota:` Se utiliza únicamente para los escenarios que empiezan la comunicación, los escenarios del llamante. Se especifica la IP:puerto donde dirigir los mensajes SIP. El valor que tome se usará únicamente para los primeros escenarios de prueba en los que no se enruta a través del Core IMS.
- `destino:` Sólo para escenarios del llamante. Es la identidad pública a la que se llame, sin incluir @dominio.
- `ruta:` Para los escenarios del llamante. Dirección IP y puerto a la que dirigir las request SIP iniciales. En los escenarios sin Core IMS se indicará la IP:puerto del Mobility Manager, y en los escenarios con Core IMS la IP:puerto del P-CSCF.
- `dominio:` Para los escenarios de registro, el dominio en que se va a registrar el usuario. Se usará open-ims.test.
- `ipLocal:` La dirección IP de la máquina desde la que se ejecuta el script.
- `puertoLocal:` El puerto de comunicación SIP que usará el script.
- `puertoLocalMedia:` Sólo se usará si se quiere especificar un puerto por el que enviar y recibir los flujos RTP de datos distinto al de por defecto, que será el primero libre a partir del 6000.
- `tasa y periodo:` Sólo para los escenarios del llamante. La tasa especificará el número de llamadas que se iniciarán de manera simultánea. Si no se especifica periodo, se lanzarán cada segundo las llamadas especificadas por el valor de tasa; si se especifica, se lanzarán las llamadas especificadas por el valor de tasa en el valor determinado por periodo (que irá en milisegundos). Ej: `-r 8` (8 llamadas por segundo), `-r 4 –rp 7000` (4 llamadas cada 7 segundos).
- `numLlamadas:` El número de llamadas que se realizarán durante la ejecución del script.
- `fichero:` Fichero del que obtener valores que se usarán en el script.

- `-rtp_echo:` Activa la función eco, por la que se hace eco de todos los paquetes RTP recibidos. Se usará para probar el Transparency Manager.

Para realizar pruebas unitarias de funcionamiento se usará la opción –m 1, sin especificar valores para tasa y periodo; de esta forma, el escenario se ejecutará una única vez. Para realizar pruebas de carga se especificará un valor superior a 1 (si se quiere parar el script, si no se ejecutaría sin fin hasta forzar la salida), y se podrán asignar valores a tasa y periodo.

A continuación, y acorde a los apartados del Capítulo 7, se irán describiendo los escenarios empleados para la realización de las pruebas, y que se entregan en el directorio "Terminales/SIPp/escenarios" del CD adjunto a la memoria. Todos los escenarios incluyen al principio un ejemplo de cómo invocarlos.

B.4.1.1 Escenarios B2BUA con conexión directa

Los escenarios del llamante utilizarán en la opción `-rsa` la dirección IP y puerto del Mobility Manager. Así mismo, indicarán como `direccionRemota` la IP:puerto donde arranquen los escenarios del llamado.

Los escenarios `uac.xml` y `uas.xml` sirven para probar un establecimiento correcto de la sesión, y la liberación desde el llamante. Los escenarios `uac2.xml` y `uas2.xml` son similares a los anteriores, pero en este caso liberando la sesión desde el llamado. Un ejemplo de ejecución de los dos primeros se muestra en la Figura 35.

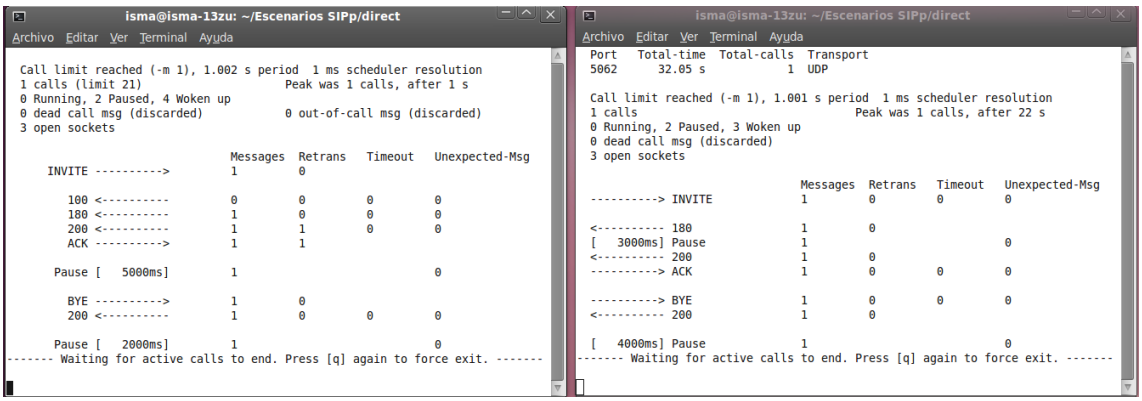

Figura 35: Escenario con conexión directa

Los escenarios `486uac.xml` y `486uas.xml` simulan el rechazo desde el llamado, y en consecuencia el fallo en el establecimiento de la sesión

Finalmente, con los escenarios `cancelUAC.xml` y `cancelUAS.xml` se puede probar el caso en el que el llamante desiste, y nuevamente falla el establecimiento de sesión.

B.4.1.2 Escenarios de prueba del Transparency Manager

Para probar que el Transparency Manager realiza bien su función, reenviando de forma correcta el flujo de datos del usuario, lo que se hará es enviar desde el escenario llamante un flujo RTP, que deberá llegar al escenario llamado pasando por el Transparency Manager. Una

vez el flujo llegue al escenario llamado, este hará eco y deberá volver al llamante, atravesando nuevamente el Transparency Manager.

El escenario `uacAudio.xml` es equivalente al escenario `uac.xml` del subapartado anterior, con la única diferencia de que hace uso de la capacidad de SIPp para enviar flujo RTP. Tras recibir el ACK de confirmación de establecimiento de sesión el llamante comienza a enviar los paquetes de audio RTP contenidos en una traza de ejemplo que se proporciona en el directorio "Terminales/SIPp/Escenarios/direct/pcap", `g711a.pcap`.

Para poder enviar tráfico RTP es necesario que SIPp se ejecute teniendo permisos de superusuario. Se pueden enviar los paquetes de cualquier flujo RTP capturado con Wireshark.

Para el llamado se ejecuta el mismo escenario `uas.xml` que en el apartado anterior, pero ejecutándolo con la opción `-rtp_echo` para que haga eco de los paquetes que reciba. En la Figura 36 se puede ver un ejemplo de ejecución, en el que se observa que el llamante ha enviado 236 paquetes de datos, y que el llamado ha hecho eco del mismo número de paquetes.

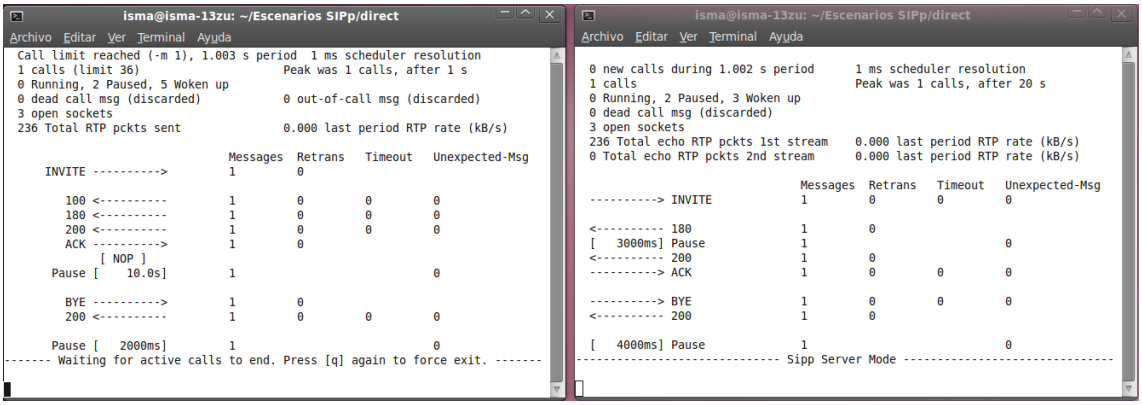

Figura 36: Escenario con conexión directa y flujo RTP

B.4.1.3 Escenarios B2BUA utilizando el Core IMS

En primer lugar es necesario registrar usuarios en el Core. Para ello se proporciona el script `register.xml`, así como el script `unregister.xml`, este último para desregistrar. Ambos escenarios se invocarán con el parámetro `-inf`, e indicando el fichero `users.csv`. Este fichero recoge los datos necesarios para poder registrar el usuario que se desee, y que serán la identidad pública, la identidad privada y la contraseña.

El fichero `users.csv`, que se proporciona en el directorio "Terminales/SIPp/Escenarios/ims" del CD adjunto, recoge los datos necesarios para poder registrar todas las identidades públicas de los usuarios que se encuentran dadas de alta en el *HSS*, y que se enumeraron en B.1.3. A la hora de registrar un usuario se lanzará el script una única vez con el parámetro `-m 1`, se escogerá a conciencia el puerto en el que se desea que el usuario intercambie la señalización SIP, y se colocará en el primer lugar del fichero `users.csv` aquella identidad que se quiera registrar. Así mismo, se utilizará la opción `-rsa` para indicar la IP:puerto del *P-CSCF*, que será a donde se mandarán los REGISTER.

Un ejemplo de ejecución del script `register.xml` puede verse en la Figura 37. Se manda una petición REGISTER a la que el Core IMS responde con un mensaje 401 Unauthorized con un reto para autenticar al usuario. Se manda un nuevo REGISTER que contenga la respuesta al reto, construida a partir de los datos del fichero `users.csv`, y el usuario queda registrado.

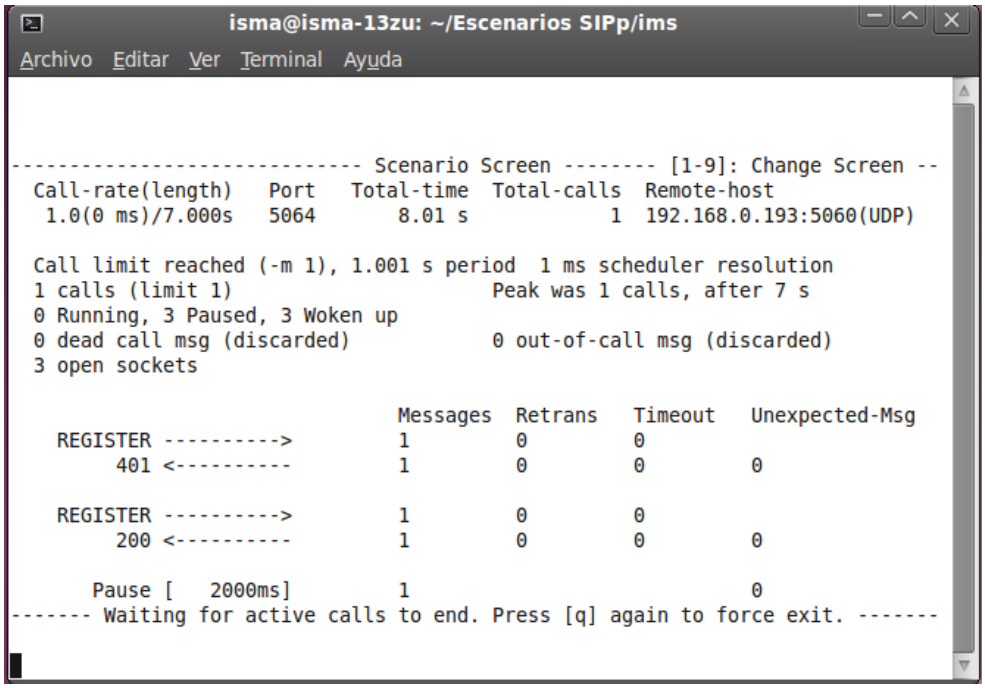

Figura 37: Escenario para registro de usuario

Los escenarios a utilizar para realizar las pruebas contienen los mismos mensajes que los vistos en los dos subapartados anteriores, pero se modifican para adaptarse al enrutamiento a través del Core IMS. Los cambios necesarios en los escenarios son los siguientes:

- Los escenarios para el llamante se invocan con el parámetro `-rsa` indicando la IP:puerto del *P-CSCF*, al igual para los escenarios de registro.
- También en los escenarios del llamante, se añade una cabecera Route indicando el *S-CSCF*. Esto forma parte del procedimiento de registro, al registrar de forma correcta una identidad se recibe una cabecera Service-Route en la respuesta 200 OK, que indica el *S-CSCF* que le dará servicio. El contenido de esta cabecera deberá incluirse en una cabecera Route cuando se envíen peticiones iniciales [2].
- Los escenarios para el llamante se invocarán con `-inf caller.csv`, indicando qué identidad se va a utilizar para el llamante. El fichero `caller.csv` se proporciona con valores para las identidades ya dadas de alta en el *HSS*. A la hora de ejecutar el escenario, se pondrá en primer lugar aquella identidad desde la que se quiera llamar.
- De forma complementaria al punto anterior, los escenarios para el llamado se invocarán con `-inf callee.csv`. Combinando distintos llamantes y llamados se podrán probar los tres tipos de movilidad descritos en 7.3. Hay que tener cuidado de ejecutar los escenarios indicando el mismo puerto SIP que se usó para el registro.
- Para los casos en que libera la sesión el llamado, SIPp no construye de forma correcta las rutas debido a un bug documentado de implementación, y pone las cabeceras Route en orden inverso al que deberían. El Core detecta que no se siguen las rutas

establecidas para el diálogo, y contesta con un 400 Bad Request – Not following indicated dialog routes. Es necesario para estos casos establecer las cabeceras Route a mano editando el fichero del escenario, y el valor es distinto según sea el escenario de movilidad que se quiera probar:

- Para el caso de movilidad en origen, `Route: <sip:mt@pcscf.open-ims.test:4060;lr>,<sip:mt@scscf.open-ims.test:6060;lr>,<sip:mo@scscf.open-ims.test:6060;lr>`
- Para el caso de movilidad en destino y de doble movilidad: `Route: <sip:mt@pcscf.open-ims.test:4060;lr>,<sip:mt@scscf.open-ims.test:6060;lr>`

Los escenarios `uac.xml` y `uas.xml` sirven para establecer una sesión, y liberarla desde el llamado. El escenario `uac2.xml` sirve para establecer una sesión, y esperar que selibere desde el llamado; con este escenario se usará `uas2Orig.xml` para el caso de movilidad en origen, y `uas2Dest.xml` para el caso de movilidad en destino o doble movilidad. Un ejemplo de ejecución de los dos primeros escenarios puede observarse en la Figura 38.

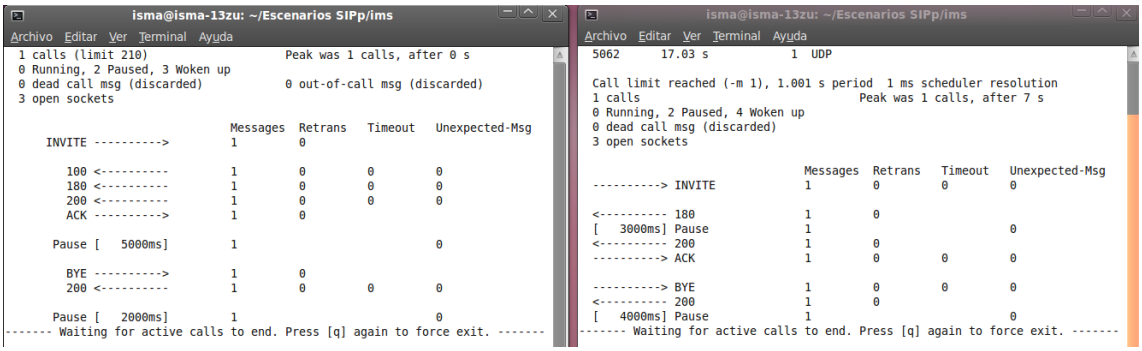

Figura 38: Escenario con sesión correcta a través del Core IMS

Los escenarios `486uac.xml` y `486.xml` sirven para probar el caso en que el llamado rechaza la sesión, y los escenarios `cancelUAC` y `cancelUAS` aquél en que el llamante desiste. Finalmente, el escenario `uacAudio.xml` se podrá usar en combinación con el escenario `uas.xml` para probar el funcionamiento del Transparency Manager. LaFigura 39 muestra un ejemplo de ejecución de este último caso.

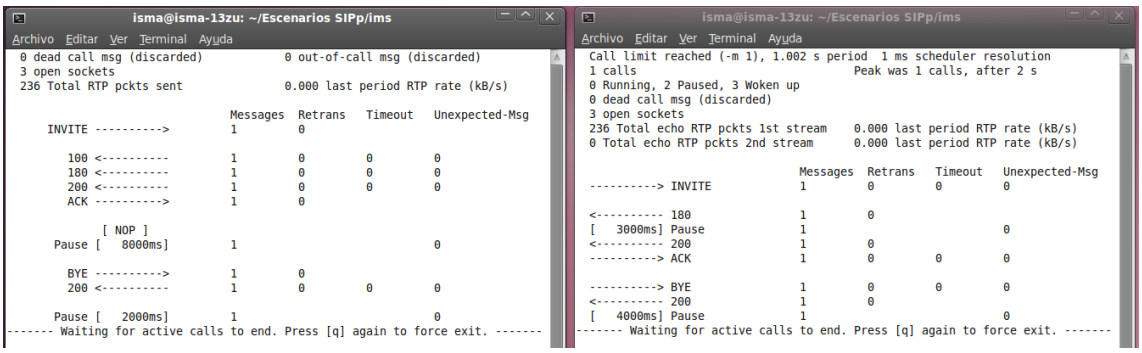

Figura 39: Escenario con flujo RTP a través del Core IMS

B.4.1.4 Escenarios de prueba de las transferencias de tipo Pull

Para probar este tipo de transferencias se utilizarán los escenarios presentados en el subapartado anterior para establecer y liberar sesiones, que utilizarán llamante y llamado, y se crearán nuevos que permitan que se produzcan las transferencias.

En primer lugar se crea un escenario para probar el único caso de error explicado en 5.1.1, aquel en el que llega al Mobility Manager un INVITE con una cabecera Replaces que no hace referencia a ningún diálogo establecido. El escenario es `pullError.xml`, que manda una cabecera Replaces con valores fijos.

Se ejecutará primeramente una combinación de escenarios para llamante y llamado que reproduzca cualquiera de los casos de movilidad, y cuando esté establecida la sesión se ejecutará el nuevo escenario. Se utilizará el parámetro `-inf callerTransfer.csv` para utilizar la identidad que se desee para el nuevo terminal, acorde al extremo en que se quiera hacer el intento de transferencia. El escenario que representa al nuevo terminal recibirá un mensaje 481 Call leg/Transaction Does Not Exist, manteniéndose activa la sesión entre llamante y llamado. A partir de aquí podrían intentarse más transferencias, o directamente liberar la sesión. Un ejemplo de ejecución puede verse en la Figura 40

, en que se intenta hacer una transferencia para el llamante.

Figura 40: Error en transferencia de tipo Pull

A continuación se crean dos nuevos escenarios, `pull.xml` y `pull2.xml`. Ambos escenarios se ejecutarán con la opción `-inf` para dos ficheros. En primer lugar se indicará el fichero con la información para poder realizar la transferencia que guarda el Mobility Manager, como se

explicó en 7.4. En segundo lugar se indicará el fichero `callerTransfer.csv`, para utilizar la identidad que se desee para el nuevo terminal simulado.

El nombre de los ficheros con la información de las transferencias seguirá el formato `call-id_incoming.csv` o `call-id_outgoing.csv`. call-id será el identificador para la sesión, y que hará referencia al call-id que utiliza el llamante. El fichero `call-id_incoming.csv` tendrá la información del diálogo que se mantiene entre el llamante y el MobilityManager, y se utilizará cuando se quieran realizar transferencias en origen (tanto para movilidad en origen como para doble movilidad). De forma complementaria el fichero `call-id_outgoing.csv` contendrá la información del diálogo que se mantiene con el llamado, y se utilizará cuando se quieran realizar transferencias en destino, para los casos de movilidad en destino y doble movilidad.

La diferencia entre los escenarios `pull.xml` y `pull2.xml` es que `pull.xml` manda el Bye para liberar la sesión, mientras que `pull2.xml` espera recibirlo. De esta forma se puede probar a que se libere la sesión desde el nuevo terminal al que se ha producido la transferencia, o que libere el otro extremo. Así mismo, con `pull2.xml` se puede realizar más de una transferencia, pues quedaría a la espera de que al producirse una nueva transferencia se le mande un Bye para terminar el diálogo SIP.

Combinando estos dos nuevos escenarios junto a los escenarios del subapartado anterior se pueden probar todas las posibles combinaciones: transferencias en el origen o en el destino, con o sin doble movilidad, más de una transferencia, probar entre medias un intento erróneo de transferencia con el escenario `pullError.xml`,…

La Figura 41 muestra un ejemplo de transferencia en el lado del llamante utilizando el escenario `pull.xml`. El llamado establece una sesión con el llamante, y posteriormente realiza una transferencia de tipo *Pull* desde otro de sus terminales. La sesión se libera desde el nuevo terminal.

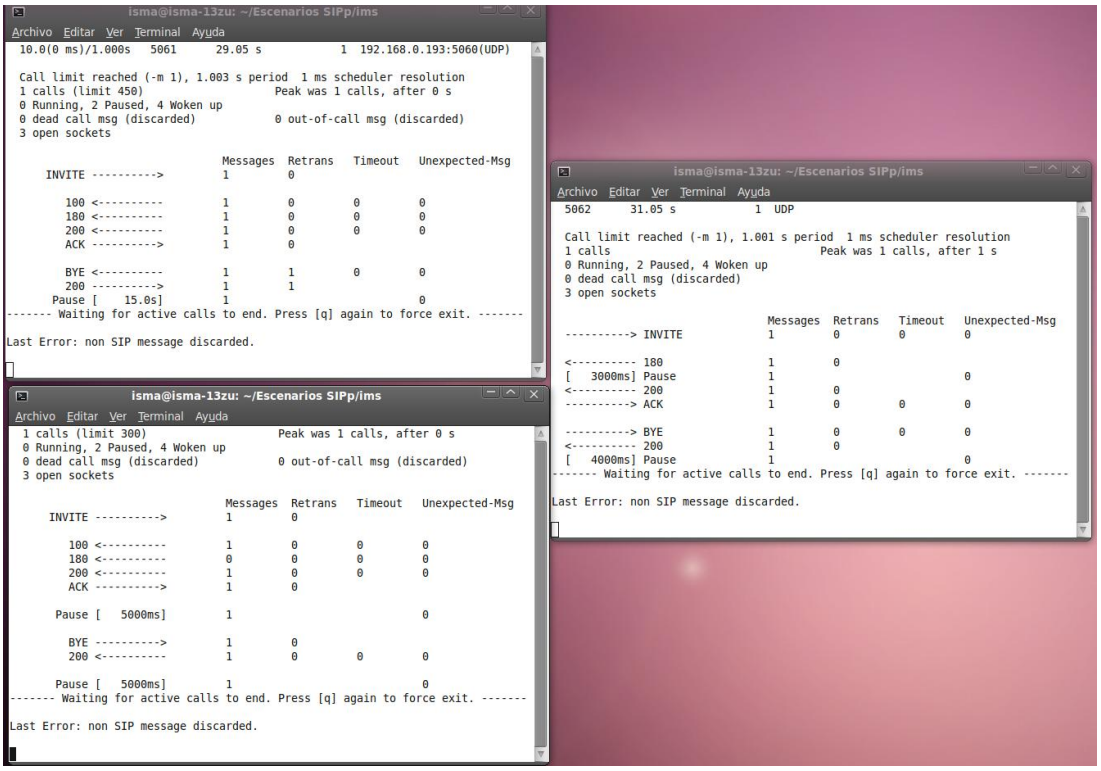

Figura 41: Transferencia de tipo *Pull* desde el llamante

La Figura 42 muestra un ejemplo de transferencia en el lado del llamado, y utilizando el escenario `pull2.xml`. El llamante establece una sesión con el llamado, y más adelante este último realiza una transferencia de tipo *Pull* hacia otro de sus terminales. La sesión se libera desde el llamante, y el llamado recibirá el BYE en el nuevo terminal.

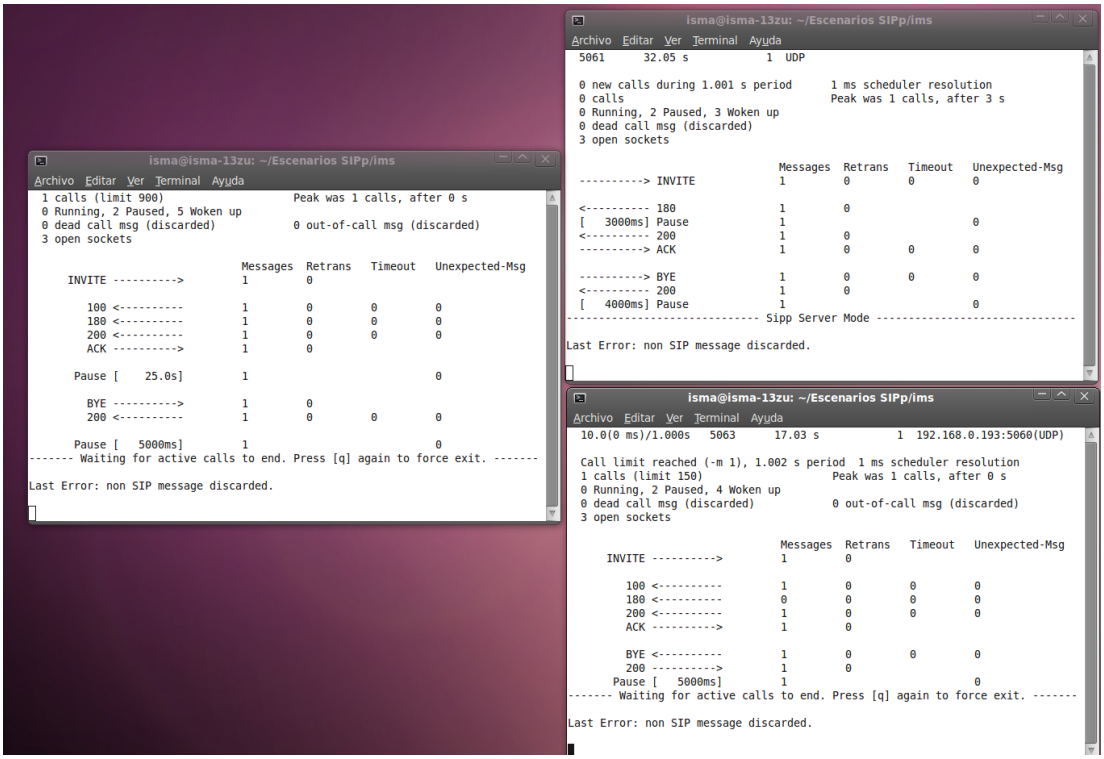

Figura 42: Transferencia de tipo *Pull* desde el llamado

Para probar la transparencia de los datos, es necesario utilizar escenarios que manden tráfico RTP. Se pueden hacer muchas pruebas, enviando tráfico sólo desde uno de los extremos y haciendo eco desde el otro, enviando desde ambos extremos y comprobando que llegan,… Se proporcionan los siguientes escenarios para probar uno de los casos y poder validar la solución:

- `uac2Audio.xml` es el llamante, y manda tráfico RTP de audio.
- Como llamado se utiliza `uas2.xml` con la opción `-rtp_echo` para que haga eco de los paquetes que recibe.
- Se realiza una transferencia en el llamante, utilizando el escenario `pullAudio.xml`, que envía también paquetes RTP.

El llamado comienza recibiendo los paquetes enviados por el primero de los terminales del llamante, y a través del Transparency Manager. Hace eco de estos paquetes, que llegan a este primer terminal del llamante también a través del TransparencyManager. Cuando se produce la transferencia y el nuevo terminal del llamante comienza a enviar paquetes, le llegan al llamado desde el mismo *endpoint* (dirección IP y puerto) del Transparency Manager. Hace eco de estos paquetes, que llegan al nuevo terminal desde el mismo *endpoint* del Transparency Manager por donde llegaban al antiguo. Un ejemplo de ejecución aparece en la Figura 43.

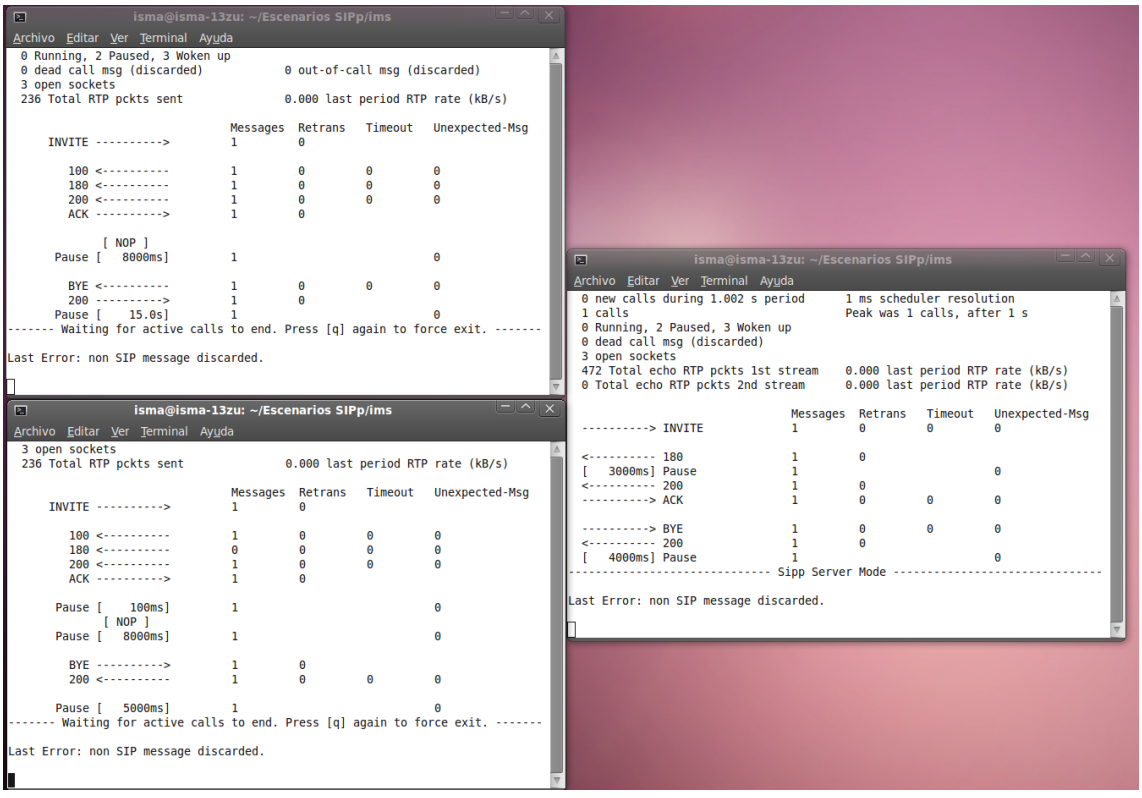

Figura 43: Transferencia de tipo *Pull* con tráfico RTP

B.4.1.5 Escenarios de prueba de las transferencias de tipo Push

Para probar este tipo de transferencias se crean cuatro nuevos escenarios:

- `uacPush.xml` y `uacPushError.xml`: Sirven para probar las transferencias en el lado del llamante. Comienzan estableciendo una sesión, y a continuación intentan realizar una transferencia de tipo *Push* enviando una petición REFER y esperando recibir los NOTIFY que informen del progreso de la transferencia. Se invocan indicando dos ficheros, `caller.csv` para la identidad a utilizar, y `transferTarget.csv` para indicar la identidad a la que realizar la transferencia.
  La diferencia entre ambos escenarios es que `uacPush.xml` espera recibir un BYE, que significaría que se ha realizado la transferencia, o que ha fallado pero se libera la sesión desde el otro extremo; mientras tanto uacPushError.xml implica que no se realiza la transferencia, y que tras un tiempo se libera la sesión desde este extremo.
- `uasPush.xml` y `uasPushError.xml`: Para probar las transferencias en el lado del llamado. Esperan a recibir un Invite para que se establezca la sesión, y a continuación intentan realizar una transferencia de la misma forma que en el caso anterior. Se invocarán en este caso con los ficheros `callee.csv` y `transferTarget.csv`.
  La diferencia entre ambos escenarios es la misma que para el caso anterior.

Para probar los casos en que falla el intento de transferencia, se utilizarán para simular el nuevo terminal objeto de la transferencia los escenarios `486.xml` y `cancelUAS.xml`. Con el primero de ellos se rechaza la transferencia, y con el segundo de ellos se cancela el intento al cumplirse el tiempo especificado sin que se acepte la transferencia. En ambos casos continúa establecida la sesión original entre llamante y llamado, y se podrían intentar nuevas transferencias o directamente liberar la sesión. La Figura 44 muestra un ejemplo de ejecución en que el objetivo de la transferencia no contesta y se cancela el nuevo diálogo SIP desde el Mobility Manager. Se utiliza el escenario `uacPushError.xml`, y se libera la sesión desde el llamante.

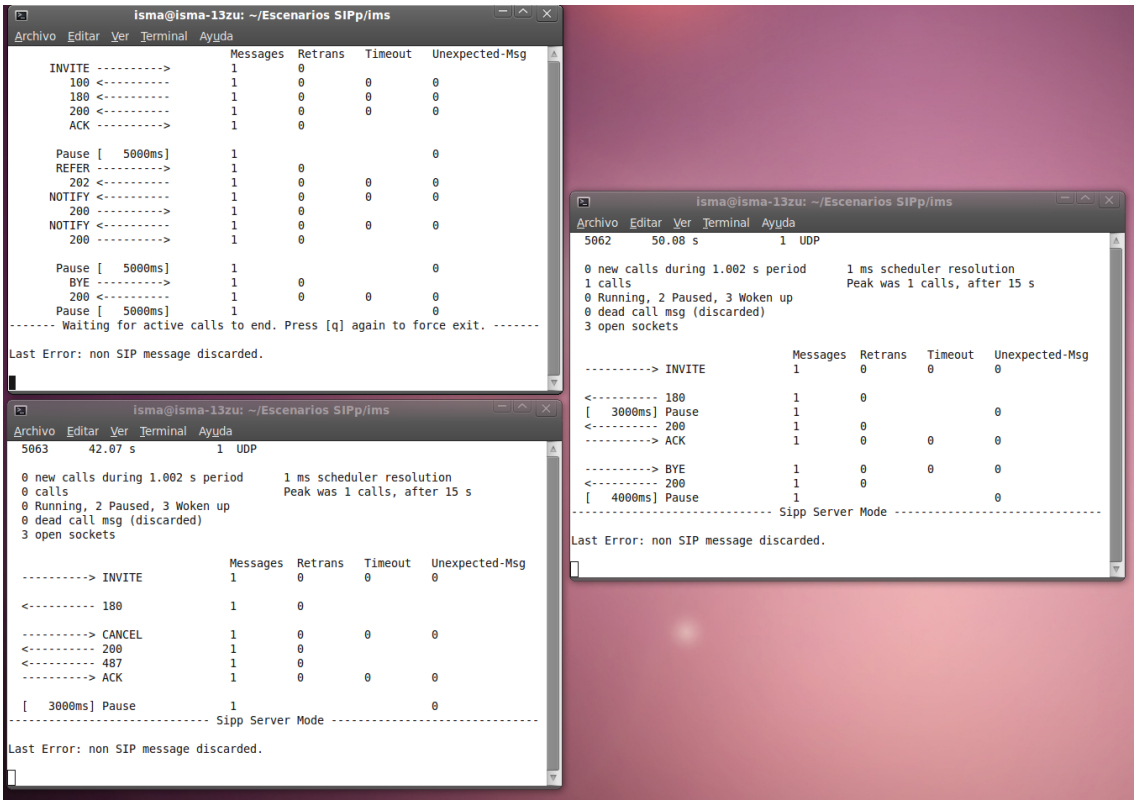

Figura 44: Error en transferencia de tipo *Push*

Para que se realicen transferencias de forma correcta, se utilizarán para simular el nuevo terminal los escenarios `uas.xml` si se espera a que se libere la sesión desde el otro extremo, y `uas2Dest.xml` si se libera desde la sesión desde el nuevo terminal. Si se quieren encadenar nuevas transferencias, se utilizarían los nuevos escenarios descritos más arriba.

La Figura 45 muestra un ejemplo de transferencia en el lado del llamante, usando el escenario `uacPush.xml`, y liberando la sesión desde el nuevo terminal utilizando el escenario `uas2Dest.xml`.

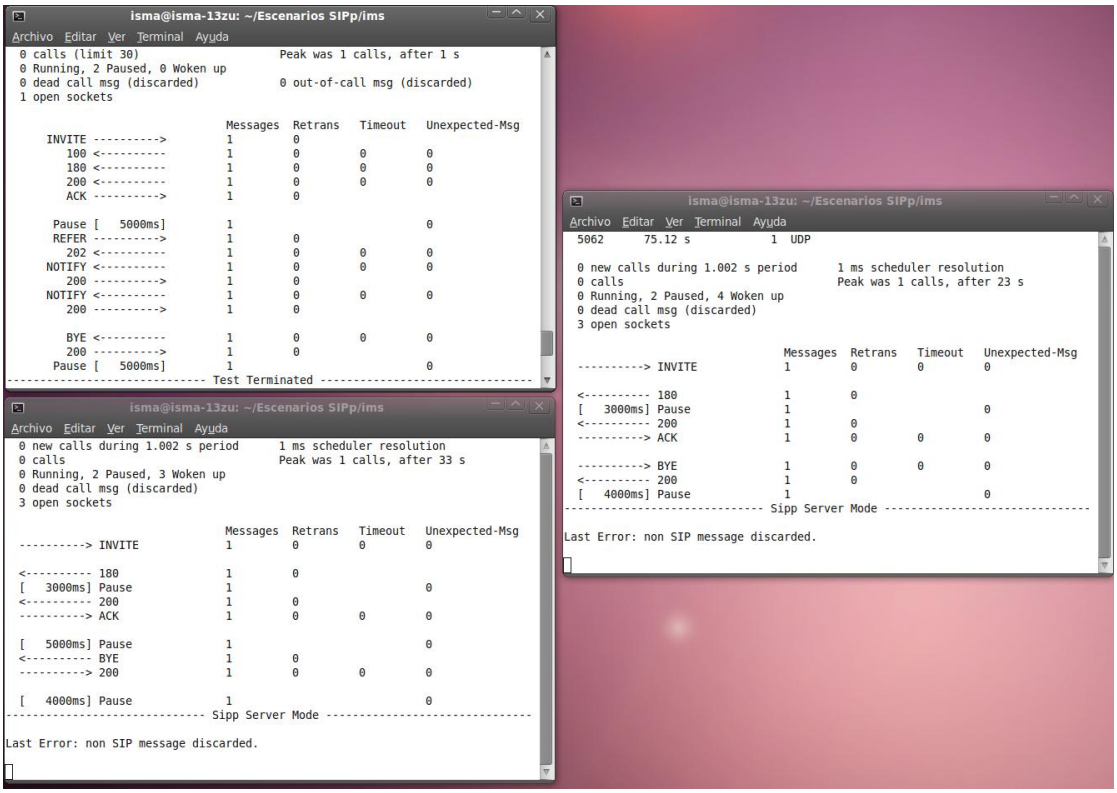

Figura 45: Transferencia de tipo *Push* desde el llamante

La Figura 46 muestra un ejemplo de transferencia en el lado del llamado, utilizando el escenario `uasPush.xml`, y esperando a que se libere la sesión desde el extremo del llamante, utilizando el escenario `uas2.xml` para el nuevo terminal.

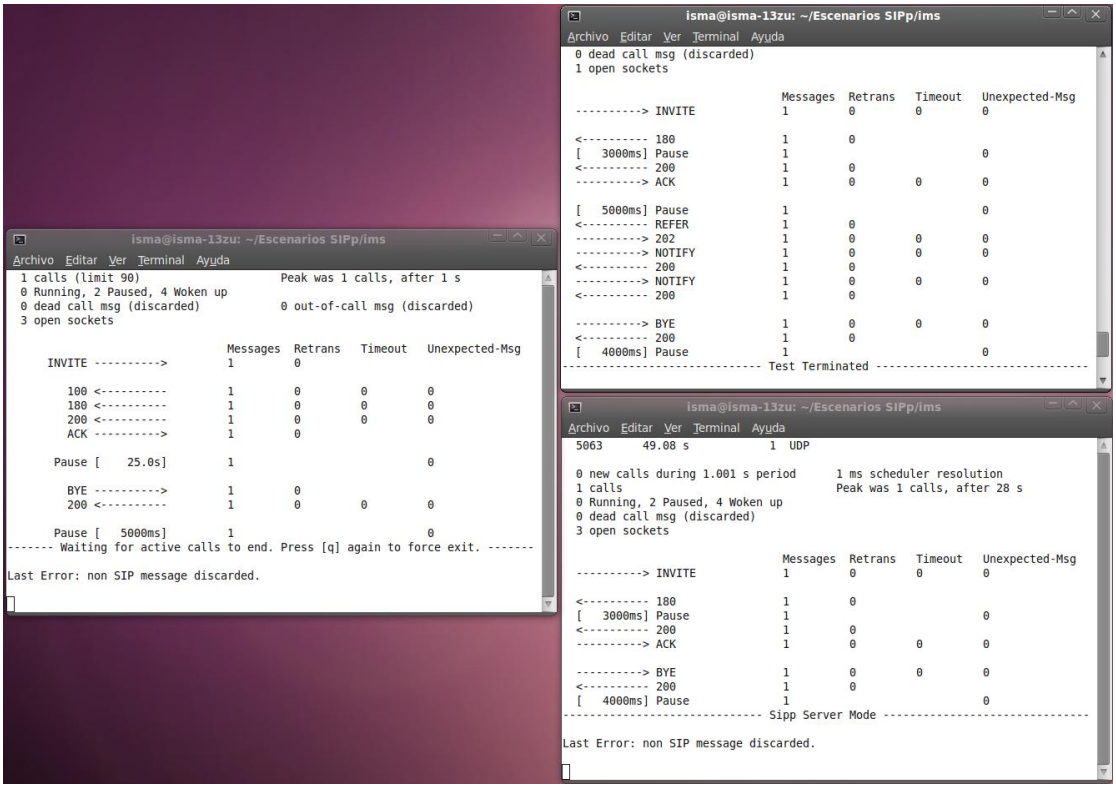

Figura 46: Transferencia de tipo *Push* desde el llamado

Al igual que para las transferencias de tipo *Pull*, hay muchas formas de probar la transparencia de los datos, siendo en este caso la forma más sencilla probar el escenario utilizando los terminales myMonster, que como ya se ha indicado en 7.5 sí permiten probar este tipo de transferencias. Para probar con escenarios SIPp se proporcionan como ejemplo los siguientes escenarios:

- `uacPushAudio.xml`: Es el primer terminal para el llamante. Manda tráfico RTP de audio, y posteriormente realiza un intento de transferencia.
- Como llamado se ejecuta el escenario `uas.xml` con la opción `-rtp_echo` para hacer eco de los paquetes recibidos.
- Como destino de la transferencia el escenario `ua2DestAudio.xml`, que manda paquetes RTP de audio y libera la sesión.

Un ejemplo de ejecución se muestra en la Figura 47. El llamado recibe en primer lugar los paquetes del primer terminal del llamante, y a través del Transparency Manager, y posteriormente los del nuevo terminal, desde el mismo endpoint del Transperency Manager. El eco de los paquetes llega primero al primer terminal, y una vez realizada la transferencia al nuevo.

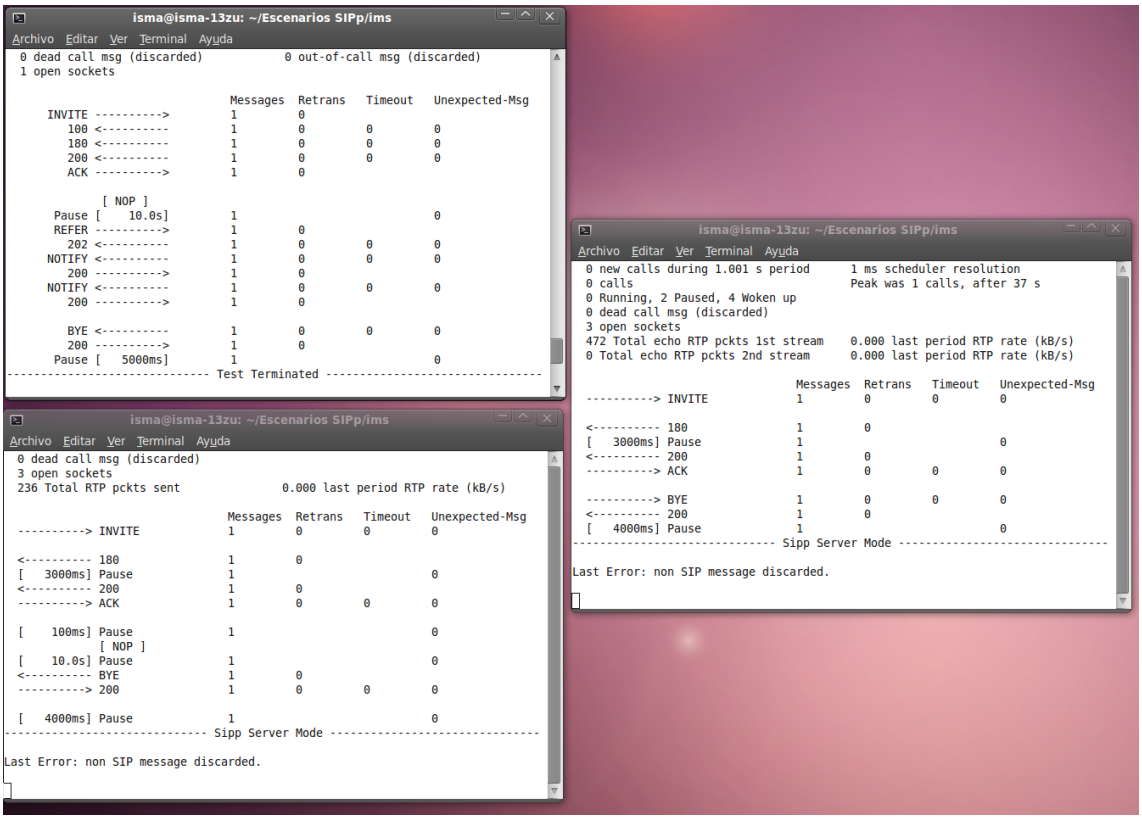

Figura 47: Transferencia de tipo *Push* con tráfico RTP

B.4.1.6 Escenarios para la medida del rendimiento

Para medir el rendimiento de la aplicación se crearon nuevos escenarios, siguiendo los requisitos de las pruebas expuestos en 7.6.

Para el llamado se crea un escenario `uasVideo.xml`, que espera a que se establezcan sesiones con él, y en ese momento empieza a mandar un flujo RTP de vídeo codificado en H263-1998.

Para probar las transferencias de tipo *Pull* se emplean los escenarios `uacVideo2.xml` y `pullVideo.xml`. El primero establece una sesión, y espera a recibir un Bye una vez realizada la transferencia. El segundo y último envía un INVITE con cabecera Replaces, y libera la sesión. Ambos escenarios guardan en un fichero la información del SDP con los puertos que utilizan, de tal forma que al establecerse la sesión invocan a *Totem*, el reproductor multimedia oficial del entorno *Gnome*. De esta forma se reproduce el vídeo que se envía desde el llamado, y de forma visual se puede comprobar el funcionamiento.

La Figura 48 y la Figura 49 muestran un ejemplo de ejecución. En la primera de ellas se tiene establecida la sesión con el primero de los terminales del llamante, mientras que en la segunda se ha realizado la transferencia y el vídeo se reproduce en el nuevo terminal.

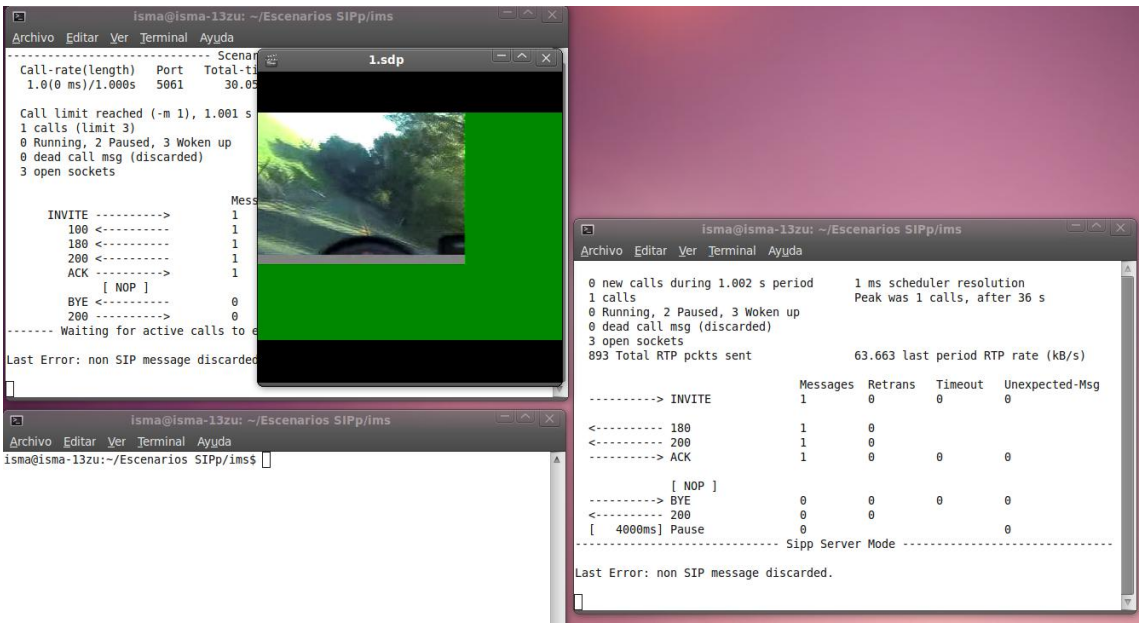

Figura 48: Transferencia de tipo *Pull*. Sesión establecida con el primer terminal

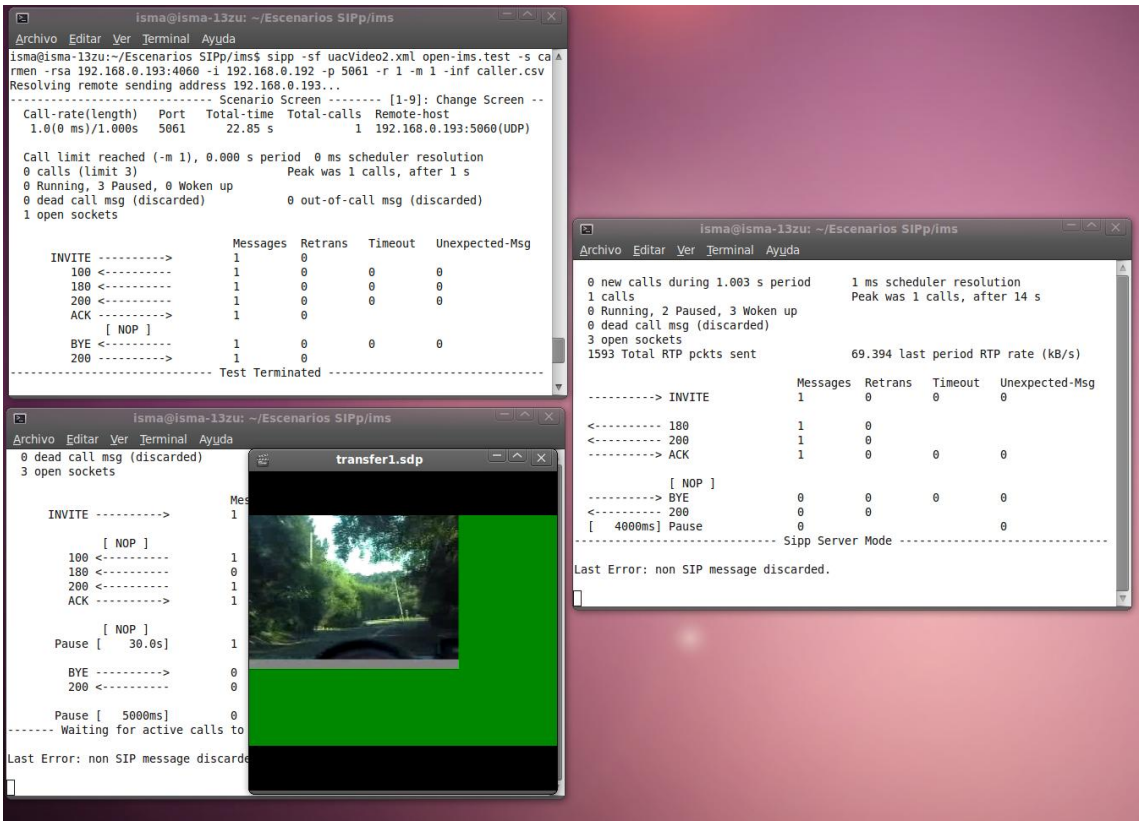

Figura 49: Transferencia de tipo *Pull*. Movilidad de sesión al segundo terminal

Para las transferencias de tipo *Push* se emplea como primer terminal el escenario `uacPushVideo.xml`, que establece una sesión y realiza una transferencia de tipo *Push* hacia un nuevo terminal. Para este nuevo terminal se emplea el escenario `uas2Video.xml`, que recibe un INVITE para establecer el nuevo diálogo con el Mobility Manager, y finalmente libera la sesión. Ambos escenarios guardan la información del SDP, y ejecutan Totem para mostrar el vídeo.

La Figura 50 y Figura 51 muestran un ejemplo de ejecución. En la primera de ellas se tiene establecida la sesión con el primero de los terminales, y en la segunda se ha producido la transferencia de la sesión hacia el nuevo terminal.

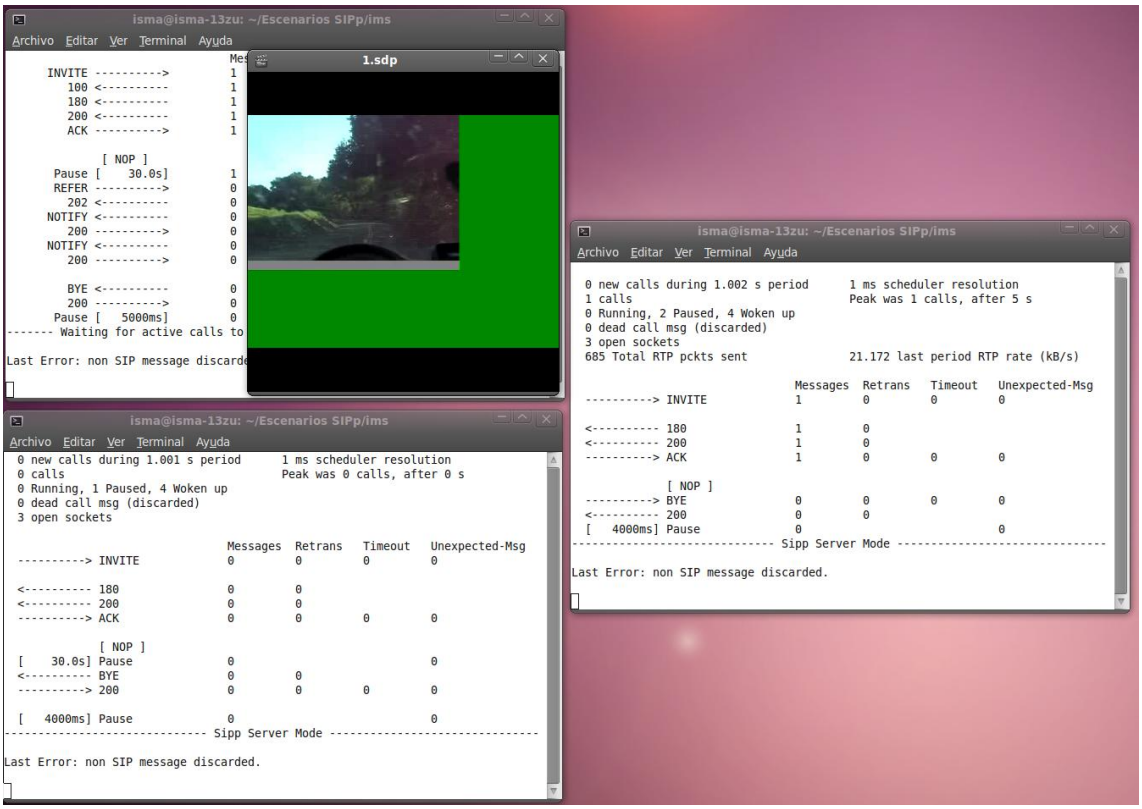

Figura 50: Transferencia de tipo *Push*. Sesión establecida con el primer terminal

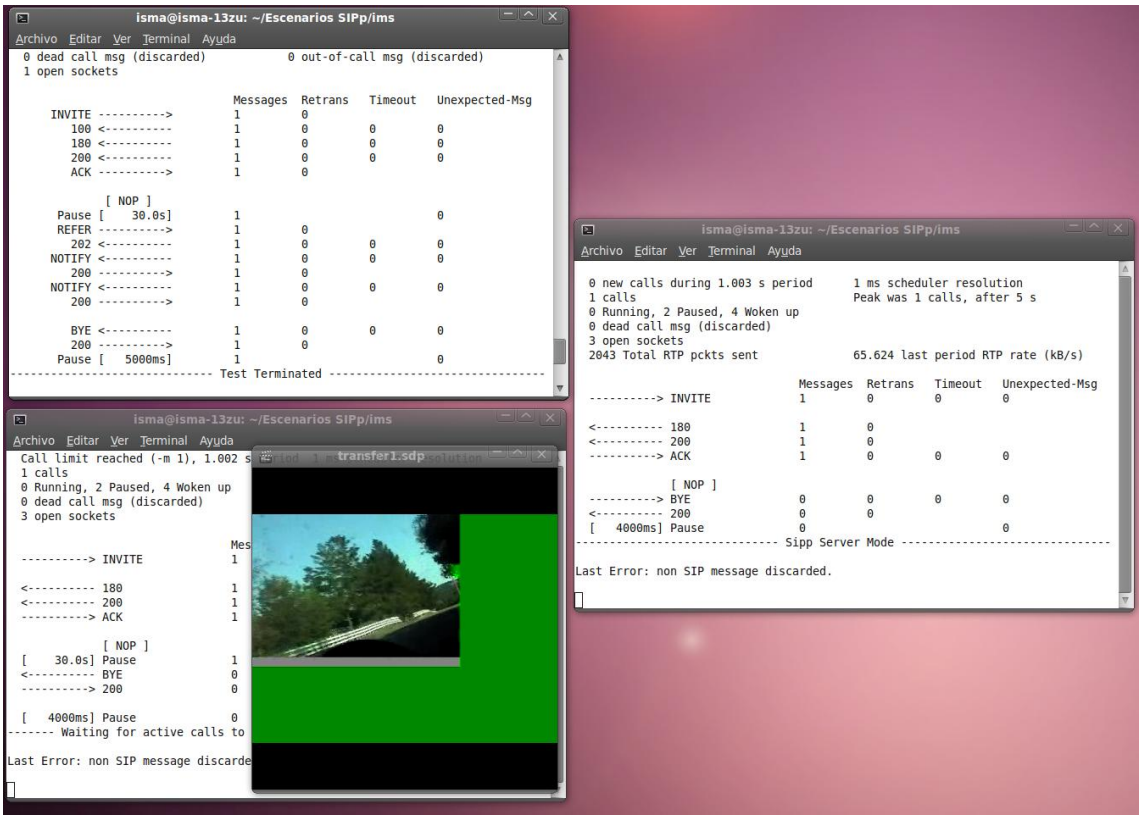

Figura 51: Transferencia de tipo *Push*. Movilidad de sesión al segundo terminal

Con estos escenarios se pueden realizar las medidas, capturando trazas y viendo los tiempos para los mensajes relevantes, y también se muestra de una forma muy visual el

comportamiento de la aplicación. El códec de vídeo que se utiliza es compatible con los *softphones* myMonster e IMSDroid, lo que permite que se puedan utilizar estos terminales para la realización de pruebas, comprobando que el vídeo se reproduce de forma correcta.

B.4.2 MyMonster

Para arrancar el terminal, una vez instalados los paquetes necesarios vistos en A.4.2 únicamente habrá que abrir un terminal en el directorio de instalación, y ejecutar el comando `$ ./monster`.

Aparecerá el interfaz del terminal, y para crear el perfil de un nuevo usuario que poder registrar en el Core IMS se pulsará el botón de Nuevo. Se rellenarán en el apartado de red IMS los datos acorde al usuario que se quiera registrar:

- Como dominio, open-ims.test.
- Display Name será el nombre que se mostrará en el terminal.
- Como Identidad Pública la que se quiera registrar, por ejemplo sip:noemi@open-ims.test.
- Como Identidad Privada la correspondiente, por ejemplo noemi@open-ims.test.
- Como clave la que se haya configurado, por ejemplo noemi.
- El *P-CSCF* será la dirección IP de la máquina virtual del Core IMS. En caso de que la máquina en que se esté ejecutando el terminal use como servidor de DNS el Core IMS, se puede poner también pcscf.open-ims.test.
- El puerto del *P-CSCF* será el 4060.
- En dirección IP se pondrá la de la máquina en que se ejecute el terminal y como puerto el que se usará para intercambiar señalización SIP, teniendo en cuenta que si la máquina es la misma que donde esté el Mobility Manager, el puerto por defecto 5060 estará ya ocupado.

La Figura 52 muestra un ejemplo con la configuración arriba comentada.

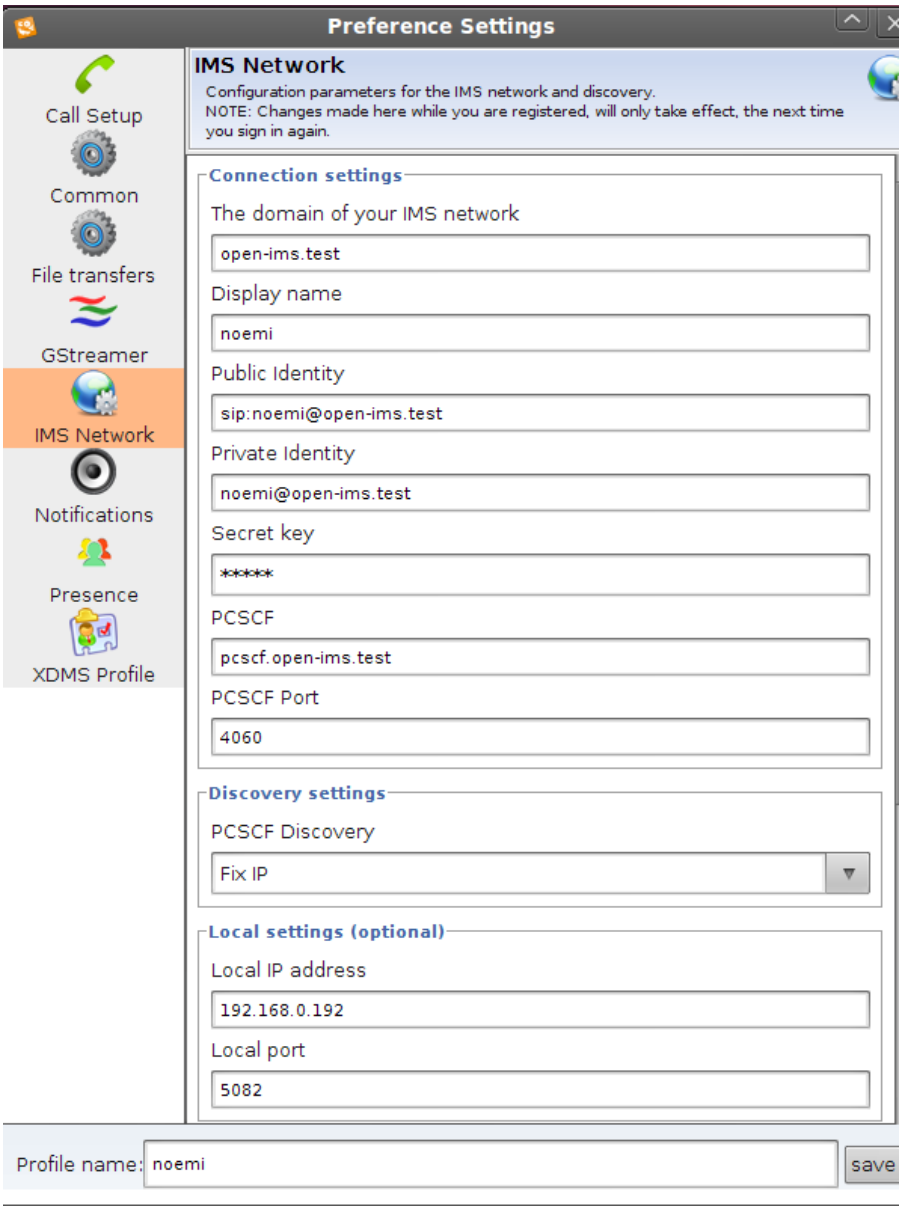

Figura 52: Configuración IMS en myMonster

En el apartado de presencia, se eliminará la opción de publicar presencia y de suscribirse a todos, ya que no se dispone de servidor de presencia en la red y los mensajes no se tratarían. La configuración se muestra en la Figura 53.

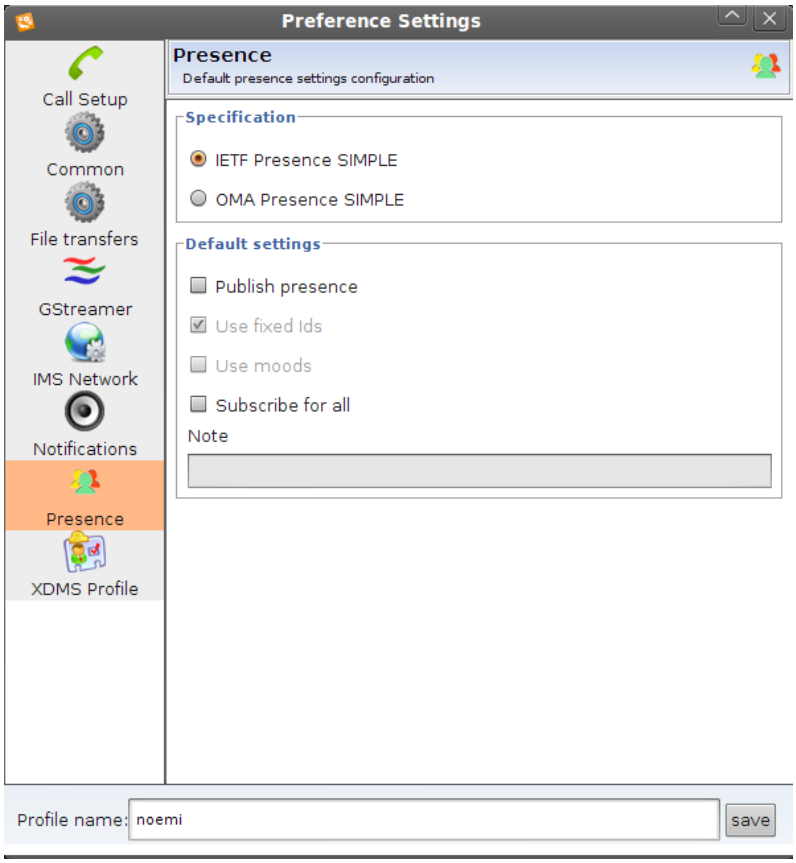

Figura 53: Configuración de presencia en myMonster

Se escogerá el nombre que se desee para el perfil y se pulsará salvar, y ya se tendrá un perfil configurado y listo para ser registrado. Se pueden configurar tantos perfiles como se desee, y arrancar más de una instancia del terminal. Para registrar el perfil únicamente habrá que seleccionarlo al iniciar el terminal, y pulsar el botón de conectar que aparece a la derecha. Para desregistrar se pulsará en la pestaña Fichero y a continuación en Desregistrar.

Para realizar una llamada, una vez se haya registrado el perfil se accederá al apartado de Llamada y se indicará la identidad pública a la que llamar, por ejemplo sip:isma@open-ims.test. A continuación se pulsará el botón de llamada o de videollamada. Si se recibe una llamada en el terminal, se mostrará una ventana emergente que permitirá el aceptarla o rechazarla.

También se pueden crear contactos, para facilitar la labor de llamar. En el apartado Contactos se hará click con el botón derecho del ratón seleccionando la opción de añadir contacto, y se rellenarán los datos. Se puede observar un ejemplo en la Figura 54. Una vez añadidos contactos, se les podrá llamar directamente desde el apartado de contactos, haciendo *click* con el botón derecho sobre el nombre del contacto y eligiendo la opción de llamar o videollamar.

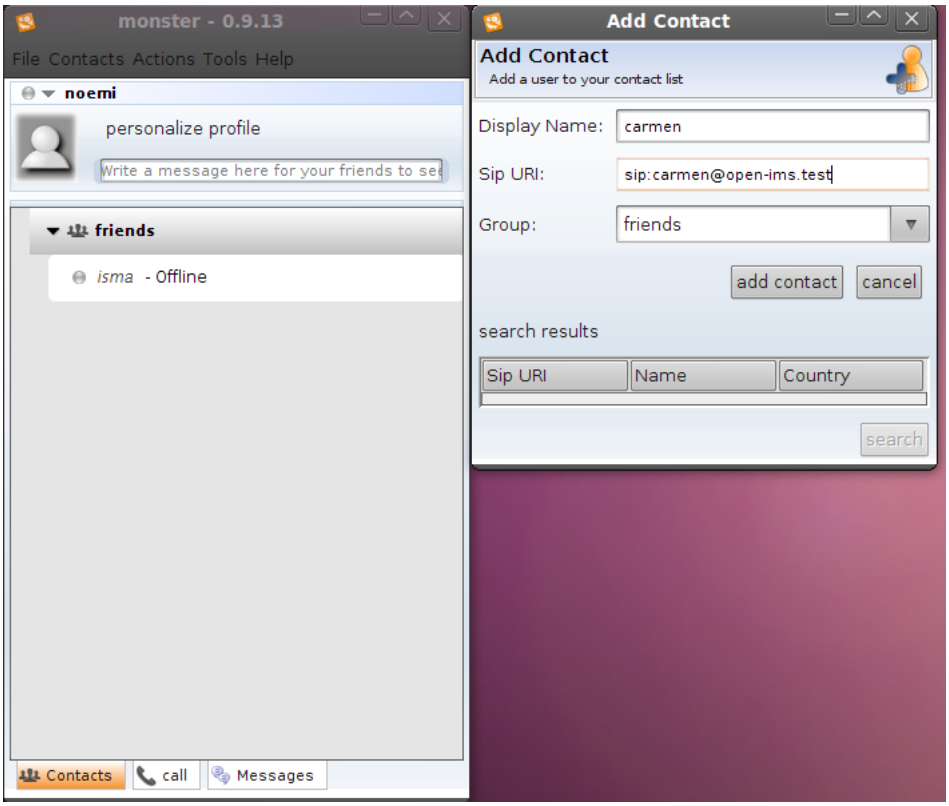

Figura 54: Añadir contacto en myMonster

B.4.3 IMSDroid

Una vez instalada la aplicación en un dispositivo Android, su configuración es muy similar al caso anterior de myMonster. Al ejecutar la aplicación se escogerá el apartado de Opciones, y ahí habrá que editar:

- En Identidad, los datos de la identidad que se quiera registrar, con el mismo significado que en el subapartado anterior. La Figura 55 muestra un ejemplo de configuración.
- En Red se configurará la dirección IP y puerto del P-CSCF. Se puede observar un ejemplo de configuración en la Figura 56.
- Finalmente en presencia se deshabilitará la opción de tenerla, por el mismo motivo que en el caso anterior.

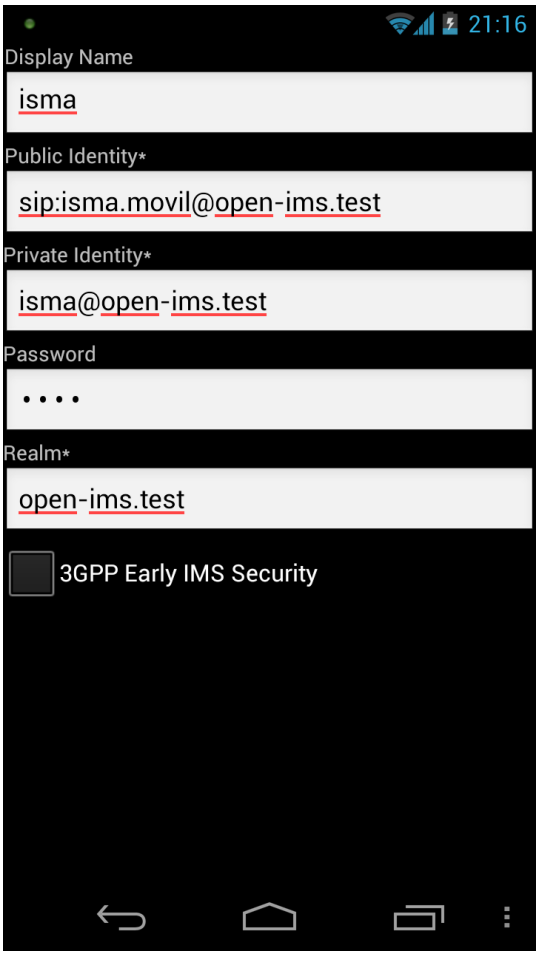

Figura 55: Configuración de identidad en IMSDroid

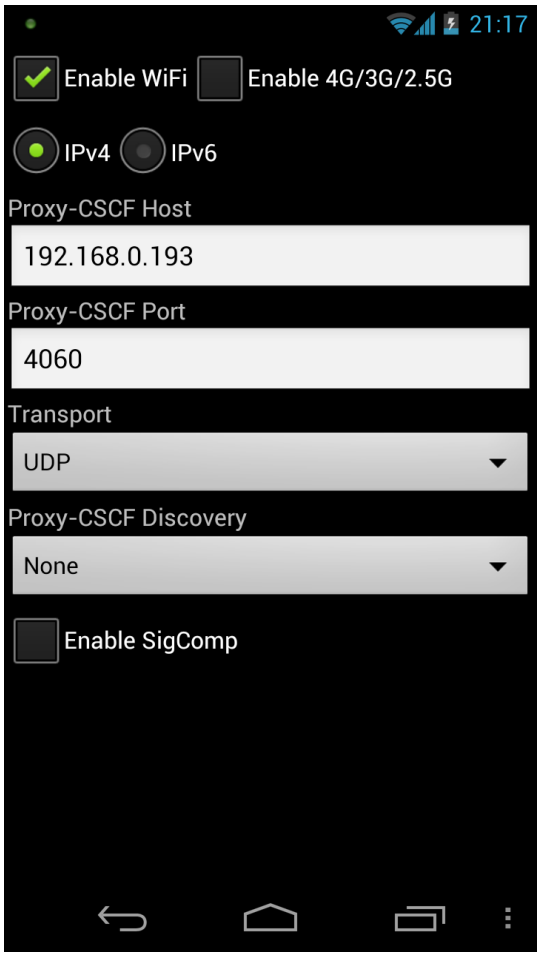

Figura 56: Configuración de red en IMSDroid

Una vez terminada la configuración, se podrá registrar la identidad en el Core y empezar a usar el terminal. Para realizar llamadas habrá que proceder como en el caso de MyMonster, escribiendo la identidad a la que se quiere llamar y pulsando llamada o videollamada. En el caso de que se reciban llamadas saltará una notificación que nos permitirá aceptarla o no.

Apéndice C

Presupuesto

Este último apéndice mostrará en primer lugar una división resumida en tareas y tiempos de lo que ha supuesto la realización del proyecto, y finalizará con un presupuesto que recoja estos tiempos junto a los equipos empleados.

C.1 División en tareas

Se realiza una división en bloques totalmente diferenciados que recogen las distintas tareas y subtareas llevadas a cabo para conseguir el objetivo. Esta separación se efectúa a efectos de evaluación ya que, tal y como se ha ido describiendo a lo largo del documento, muchas de estas tareas se han ido realizando de forma gradual y en paralelo.

El tiempo se expresa en jornadas de trabajo de 8 horas, y el resultado puede observarse en la Tabla 6.

Tareas y subtareas	Duración
Documentación y revisión de fuentes bibliográficas	**16**
Core IMS	**8**
Instalacion	3
Configuración del servicio de movilidad y de usuarios	5
Mobility Manager	**33**
Desarrollo del B2BUA SBB	12
Desarrollo del Transparency SBB	5
Desarrollo del Pull Transfer SBB	8
Desarrollo del Push Transfer SBB	8
Transparency Manager	**14**
Desarrollo inicial con click	9
Modificación para uso de sockets	5
Terminales	**9**
Creación de escenarios SIPp	7
Manejo de myMonster	1
Manejo de IMSDroid	1
Pruebas	**14**
Elaboración de la memoria	**35**
TOTAL	**129**

Tabla 6: División en tareas y subtareas

Considerando 22 jornadas de trabajo al mes, el tiempo para la realización del proyecto ha sido de 5 meses y 19 jornadas adicionales.

C.2 Cálculo de presupuesto

A continuación se detalla un presupuesto para el proyecto, y se incluye en él el coste de dos Ingenieros Sénior que han ayudado en su realización. María Calderón en la tarea de documentación y en la de elaboración de la memoria, y Víctor Sandonís en las tareas del Transparency Manager y las pruebas. Su tiempo dedicado es adicional al reflejado en el subapartado anterior, que únicamente hace referencia al Ingeniero Proyectista.

PRESUPUESTO DE PROYECTO

1.- Autor: Ismael Fernández Castellano

2.- Departamento: Ingeniería Telemática

3.- Descripción del Proyecto:

- Titulo **Movilidad de sesiones sobre plataforma IMS**
- Duración (meses) **6**

Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):

Euros 24000

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Fernández Castellano, Ismael	Ingeniero	5,9	2.694,39	15.896,90	
Sandonís Consuegra, Víctor	Ingeniero Senior	0,5	4.289,54	2.144,77	
Calderón Pastor, María	Ingeniero Senior	0,3	4.289,54	1.286,86	
				0,00	
				0,00	
		Hombres mes 6,7		**Total**	19.328,53

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1.575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
Ordenador de desarrollo y pruebas	800,00	100	5	60	70,67
Ordenador de pruebas	700,00	100	3	60	35,00
Terminal Android	350,00	100	1	60	5,83
		100		60	0,00
		100		60	0,00
					0,00
				Total	111,50

[d)] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
	Total	0,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	19.329
Amortización	112
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	3.888
Total	23.328

Referencias

[1] 3GPP TS 23.228: “IP Multimedia Subsystem (IMS); Stage 2”.

[2] 3GPP TS 24.229: “IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3”.

[3] 3GPP TS 29.229: “Cx and Dx interfaces based on the Diameter protocol; Protocol details”.

[4] 3GPP TS 29.329: “Sh interface based on the Diameter protocol; Protocol details”.

[5] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., y E. Schooler, “SIP: Session Initiation Protocol”, RFC 3261, Junio de 2002.

[6] Mahy, R., Biggs, B., y R. Dean, “The Session Initiation Protocol (SIP) “Replaces” Header”, RFC 3891, Septiembre de 2004.

[7] Rosenberg, J., y H. Schulzrinne, “An Offer/Answer Model with the Session Description Protocol (SDP)”, RFC 3264, Junio de 2002.

[8] Okumura, S., Sawada, T., and P. Kyzivat, “Session Initiation Protocol (SIP) Usage of the Offer/Answer Model”, RFC 6337, Agosto de 2011.

[9] Sparks, R., Johnston, A., y D. Petrie, “Session Initiation Protocol (SIP) Call Control – Transfer”, RFC 5589, Junio de 2009.

[10] Sparks, R., “The Session Initiation Protocol (SIP) Refer Method”, RFC 3515, Abril de 2003.

[11] Roach, A. B., “Session Initiation Protocol (SIP) – Specific Event Notification”, RFC 3265, Junio de 2002.

[12] Handley, M., Jacobson, B., y C.Perkins, “SDP: Session Description Protocol”, Julio de 2006.

[13] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, Julio de 2003.

[14] “Fraunhofer Institute for Open Communication Systems”, http://www.fokus.fraunhofer.de/en/fokus/index.html.

[15] “Open IMS Core”, http://www.openimscore.org/.

[16] “JAIN SLEE (JSLEE) 1.1 Specification, Final Release”, http://www.jcp.org/en/jsr/detail?id=240.

[17] “Iniciativa JAIN”, http://java.sun.com/products/jain/.

[18] “Mobicents”, http://www.mobicents.org/index.html.

[19] “Java RMI. Remote Method Invocation Home”, http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html.

[20] “SIPp”, http://sipp.sourceforge.net/.

[21] “MyMONSTER”, http://www.monster-the-client.org/.

[22] “IMSDroid”, http://code.google.com/p/imsdroid/.

[23] “The Click Modular Router Project”, http://www.read.cs.ucla.edu/click/.

[24] “Eclipse IDE”, http://www.eclipse.org/.

[25] “Wireshark”, http://www.wireshark.org/.

[26] Elliot Rusty Harold, “Java Network Programming, 3rd Edition”, Ed. O’Reilly, Octubre 2004.

[27] “VMware Player”, http://www.vmware.com/products/player/.

[28] “Variables de entorno en Ubuntu”, https://help.ubuntu.com/community/EnvironmentVariables.

[29] “GNU Screen”, http://www.gnu.org/software/screen/.

[30] Víctor Sandonís, Ismael Fernández e Ignacio Soto, “SIP-Based Context-Aware Mobility for IPTV IMS Services”, EuroITV2012, http://www.euroitv2012.org/AdjProc_EuroITV2012.pdf.